SKY DIAMOND PAU V1.12 AUDIT

13-04-2026 - 17-04-2026

Table of contents

1. Project brief

Title	Description
Client	Spark
Project name	Sky Diamond PAU v1.12 Audit
Timeline	13-04-2026 - 17-04-2026

Project Log

Date	Commit Hash	Note
11-04-2026	fea256adfc90d162c33917099985a69427a1129d	**audit commit (release v1.12.0-beta.0)**
24-04-2026	dc54647a8fb8cd9fe313e95d149cb9c65d36e2c6	**reaudit commit (release v1.12.0-rc.0)**
06-05-2026	fc2fe7fe5991b4f84f39ec88239b5ae4ffaacc8f	**reaudit commit (release v1.12.0-rc.1)**
14-05-2026	ebcc56d45d1835d74915c2446ecc71cb61589c5f	**reaudit commit (release v1.12.0-rc.2)**
25-05-2026	c741e32c55ceb4e4bc4245e89cc057aa6fb34e3a	**reaudit commit (release v1.12.0-rc.3)**
27-05-2026	a84fe9616412e3b6d10ea64301a3481f7dcf9a05	**post audit commit (release v1.12)**

Short Overview

Together with Spark, Unvariant performed a focused security review of stage 2 of Spark's migration to a diamond architecture from 13-04-2026 to 17-04-2026.

This report covers the issues identified during that process and the remediation status confirmed with the Spark team.

Project Scope

The audit covered the following files:

- ALMProxy.sol
- ALMProxyFreezable.sol
- AccessControls.sol
- Beacon.sol
- Controller.sol
- ControllerSharedStorage.sol
- PAUFactory.sol
- RateLimits.sol
- IFacetBase.sol
- FacetBase.sol
- IAaveFacet.sol
- AaveFacet.sol
- IBasinFacet.sol
- BasinFacet.sol
- ICCTPFacet.sol
- CCTPFacet.sol
- ICentrifugeFacet.sol
- CentrifugeFacet.sol
- ICurveFacet.sol
- CurveFacet.sol
- IDAIUSDSFacet.sol
- DAIUSDSFacet.sol
- IERC4626Facet.sol
- ERC4626Facet.sol
- IERC7540Facet.sol
- ERC7540Facet.sol
- IFarmFacet.sol
- FarmFacet.sol
- ILayerZeroFacet.sol
- LayerZeroFacet.sol
- IMapleFacet.sol
- MapleFacet.sol
- IMerklFacet.sol
- MerklFacet.sol
- IOTCBuffer.sol
- OTCBuffer.sol
- IOTCFacet.sol
- OTCFacet.sol
- IPendleFacet.sol
- PendleFacet.sol
- IPSMFacet.sol
- PSMFacet.sol
- IPSM3Facet.sol
- PSM3Facet.sol
- ISparkVaultFacet.sol
- SparkVaultFacet.sol
- ISuperstateFacet.sol
- SuperstateFacet.sol
- ITransferAssetFacet.sol
- TransferAssetFacet.sol
- IUniswapV3Facet.sol
- UniswapV3Facet.sol
- UniswapV3Utils.sol
- IUniswapV4Facet.sol
- UniswapV4Facet.sol
- IUSDEFacet.sol
- USDEFacet.sol
- IUSDSFacet.sol
- USDSFacet.sol
- IWEETHFacet.sol
- WEETHFacet.sol
- IWEETHModule.sol
- WEETHModule.sol
- IWrapProxyETHFacet.sol
- WrapProxyETHFacet.sol
- IWSTETHFacet.sol
- WSTETHFacet.sol
- ApproveLib.sol
- IALMProxy.sol
- IALMProxyFreezable.sol
- IAccessControls.sol
- IBeacon.sol
- IController.sol
- IEnumerableIntegrations.sol
- IPAUFactory.sol
- IRateLimits.sol
- RateLimitHelpers.sol

2. Finding severity breakdown

All vulnerabilities identified in the audit are grouped by their potential impact using the following severity scale:

Severity	Description
Critical	Vulnerabilities that enable theft of funds, lock user access to assets, or otherwise cause funds to be moved away from their rightful owners.
High	Bugs that can halt contract operation and typically require manual state intervention or a contract replacement to recover.
Medium	Bugs that disrupt intended logic or open DoS vectors but do not directly lead to loss of funds.
Low	Low-impact concerns with no immediate material effect that are straightforward to correct.
Informational	Best-practice recommendations and quality improvements with negligible security impact.

After the Client reviews the findings reported by Unvariant, each item receives one of the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Client is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Medium	2 (2 fixed, 0 acknowledged)
Low	8 (6 fixed, 2 acknowledged)
Informational	9 (6 fixed, 3 acknowledged)
Total	19 (14 fixed, 5 acknowledged)

4. Conclusion

A total of 19 findings were identified during this review. Each finding was reviewed by the Spark team and either acknowledged with supporting comments or remediated.

Additionally, some findings discovered by Octane were shared by the Spark team pre-audit. Out of many potential issues we identified 3 true positives and researched each of them further.

These are MEDIUM-2, LOW-5 and LOW-6.

Remediations were implemented in commit [a84fe9616412e3b6d10ea64301a3481f7dcf9a05](). Unvariant reviewed these changes and did not identify new issues introduced by the fixes.

Unvariant considers this release to be safe and ready for deployment.

5. Findings report

MEDIUM-01	**Compromised `weETH` may allow draining ALM proxy via dynamic `eETH` address lookup**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Lines:

- WEETHFacet.sol#L105
- WEETHFacet.sol#L142

`WEETHFacet.deposit()` and `WEETHFacet.requestWithdraw()` resolve the `eETH` token and `liquidityPool` addresses dynamically by reading them from the external `weETH` contract on every call, and then pass them to `ApproveLib.approve()` as the `token` argument. The `weETH` contract is a proxy contract. If `weETH` is compromised and upgraded to return attacker-chosen values from `eETH()`, the facet may issue an allowance from the ALM proxy for an arbitrary token to an arbitrary spender.

```
function deposit(uint256 amount, uint256 minSharesOut) external ... {
    ...

    address eeth = IWEETHLike(weeth).eETH(); // <- can be an arbitrary address
    ...
    uint256 eethAmount = ILiquidityPoolLike(liquidityPool).amountForShare(eethShares);
    // ^- can be an arbitrary amount

    ApproveLib.approve(eeth, proxy, weeth, eethAmount);
    // ^- arbitrary eeth and arbitrary eethAmount
    ...
}
```

A compromise of the external `weETH` contract may allow draining arbitrary ERC-20 balances held by the ALM proxy in arbitrary amounts, bypassing the `LIMIT_DEPOSIT` and `LIMIT_REQUEST_WITHDRAW` rate limits entirely. This expands the trust surface of the `weETH` integration beyond `eETH`/`weETH` positions to every token held by the proxy.

Recommendation

We recommend resolving the `eETH` token address once at construction and storing it as an `immutable`, so that the facet no longer depends on `weETH` returning a trustworthy value on every interaction.

```
contract WEETHFacet is IWEETHFacet, FacetBase {
  ...
+ address public immutable eeth;

  constructor(address weeth_, address weth_) {
    ...
+   eeth  = IWEETHLike(weeth_).eETH();
  }
  ...
  function deposit(uint256 amount, uint256 minSharesOut) external ... {
      ...
-     address eeth = IWEETHLike(weeth).eETH();
      ...
  }
  ...
  function requestWithdraw(address weethModule, uint256 weethShares, uint256
minEETHShares) ... {
      ...
-     address eeth = IWEETHLike(weeth).eETH();
      ...
  }
  ...
}
```

MEDIUM-02	**Basin deposit/withdraw lack slippage protection**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Lines:

- BasinFacet.sol#L43
- BasinFacet.sol#L69

`BasinFacet.deposit()` and `BasinFacet.withdraw()` forward calls to Basin and unconditionally accept the returned share or asset amounts with no minimum-out validation. Unlike `ERC4626Facet`, which includes `minSharesOut` / `minAssetsOut` parameters, these functions carry no equivalent guard. Rate limits constrain nominal throughput but not conversion quality. Under the threat model where `RELAYER_ROLE` may be compromised, an attacker can trigger deposits or withdrawals during periods of unfavorable Basin rates such as deposit fees, exit penalties, or partial fills, and the facet accepts those outcomes unconditionally.

Impact: A compromised relayer can repeatedly trigger deposits and withdrawals at unfavorable exchange rates, extracting avoidable principal loss bounded only by the configured rate-limit budget.

Recommendation

We recommend adding `minSharesOut` to `BasinFacet.deposit()` and `minAssetsOut` to `BasinFacet.withdraw()` as relayer-supplied parameters, reverting if the returned amount falls below the specified threshold, consistent with the pattern already used in `ERC4626Facet`.

LOW-01	**Reentrancy guard uses non-fixed storage slot**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Lines:

- Controller.sol#L16
- FacetBase.sol#L12

Both `Controller` and `FacetBase` inherit OpenZeppelin v5's `ReentrancyGuard`, which stores its `_status` flag in a compiler-assigned storage slot (slot 0) rather than an ERC-7201 namespaced slot. This currently functions correctly because `ReentrancyGuard._status` is the only non-namespaced variable in the entire inheritance chain of both `Controller` and `FacetBase`, so slot 0 is consistently mapped to the reentrancy status across all execution contexts. However, if a future facet inherits from an additional contract that uses a compiler-assigned storage slot and appears before `ReentrancyGuard` in the inheritance linearization, `_status` would shift to a different slot. The facet would then read and write the reentrancy flag at a slot that differs from the one used by `Controller` and other facets, breaking the reentrancy guard.

Recommendation

We recommend replacing the inherited OpenZeppelin `ReentrancyGuard` with a custom implementation that stores `_status` at a fixed ERC-7201 namespaced slot.

LOW-02	**Relayer removal emits event regardless of actual state change**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Lines:

- AccessControls.sol#L37
- ALMProxyFreezable.sol#L36

The `AccessControls.removeRelayer()` and `ALMProxyFreezable.removeRelayer()` functions call `AccessControl._revokeRole()` and unconditionally emit `RelayerRemoved`. OpenZeppelin's `AccessControl._revokeRole()` returns a `bool` indicating whether the role was actually revoked and it returns `false` if the account did not have the role. Since the return value is not checked, the `RelayerRemoved` event is emitted even when the provided address was not a relayer, which may produce misleading off-chain data.

Recommendation

We recommend checking the return value of `AccessControl._revokeRole()` and emitting the event only when the role was actually revoked:

```
  function removeRelayer(address relayer) external override nonReentrant
onlyRole(FREEZER_ROLE) {
-     _revokeRole(RELAYER_ROLE, relayer);
-     emit RelayerRemoved(relayer);
+     if (_revokeRole(RELAYER_ROLE, relayer)) {
+         emit RelayerRemoved(relayer);
+     }
  }
```

LOW-03	**Uncached external view call result may return inconsistent values**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

For example, in `ERC4626Facet.deposit()`, the external view function `IERC4626Like(token).asset()` is called twice with a state-changing operation in between:

1. First call: used to compute the rate limit key.
2. State change: `_decreaseRateLimit` modifies the rate limits contract state.
3. Second call: used to determine which token to approve for spending by the proxy.

```
function deposit(address token, uint256 amount, uint256 minSharesOut) external override
nonReentrant onlyRole(RELAYER_ROLE) returns (uint256 shares)
{
        // 1. First asset() call — inside _getDepositRateLimitKey
        _decreaseRateLimit(_getDepositRateLimitKey(token), amount);

        address proxy = _getSharedControllerStorage().proxy;

        // 2. Second asset() call — for the approval
        ApproveLib.approve(IERC4626Like(token).asset(), proxy, token, amount);

        ...

}
```

If the `token` contract is compromised, its `asset()` function can return different addresses on the first and second call within the same transaction, despite being declared as `view`.

Attack mechanism

A `view` function called via `STATICCALL` cannot modify state, but it can read external state. Since `_decreaseRateLimit` modifies the rate limits contract's storage between the two `asset()` calls, a malicious `asset()` implementation can detect this change and return different values:

```
// Example of malicious token contract

contract MaliciousVault {

        address immutable rateLimits;
        bytes32 immutable rateLimitKey;
        address immutable legitimateAsset;
        address immutable targetAsset;

        function asset() external view returns (address) {

                // Read rate limit state
                uint256 currentLimit =
IRateLimits(rateLimits).currentRateLimit(rateLimitKey);

                if (currentLimit == expectedOriginalValue) {
                        // First call: rate limit not yet decreased
                        return legitimateAsset;
                } else {
                        // Second call: rate limit already decreased
                        return targetAsset;
                }
        }
}
```

A compromised `token` contract can exploit this to steal funds via decimals mismatch: the `amount` parameter passed to `ApproveLib.approve` was intended for the legitimate asset's decimal precision. When the approval is issued for a token with fewer decimals, the same numeric `amount` represents a significantly larger value.

Recommendation

We recommend caching the results of external view calls in local variables and reuse them throughout the function instead of making repeated calls. This pattern should be applied across all facets where the same external view function is called multiple times.

LOW-04	**LayerZeroFacet.quoteTransfer() allows unrestricted call from ALM proxy**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Line: LayerZeroFacet.sol#L187

In `LayerZeroFacet.quoteTransfer()`, the `quoteSend()` function on the OFT contract is called via `IALMProxy.doCall()`:

```
fee = abi.decode(
    IALMProxy(_getSharedControllerStorage().proxy).doCall(
        oft,
        abi.encodeCall(ILayerZeroLike.quoteSend, (sendParams, false))
    ),
    (MessagingFee)
);
```

`ILayerZeroLike.quoteSend()` is a `view` function, meaning it should not modify state. However, `doCall()` uses `Address.functionCall()`, which performs a regular `call` rather than a `staticcall`. The `LayerZeroFacet.quoteTransfer()` is a `public` function with no access control, any caller can pass an arbitrary `oft` address. The ALM proxy then executes a regular `call` to that address on behalf of itself, which may allow triggering state-changing logic on arbitrary contracts with the proxy as a sender. While this is practically unlikely for other contracts to have state-changing `quoteSend()` function, it is also possible that some contracts have `fallback()` function with some state changing logic.

Recommendation

We recommend using `staticcall` instead of `call` when invoking view functions. This could be achieved by adding a `doStaticCall()` method to `ALMProxy` and updating `quoteTransfer()` accordingly:

```
- IALMProxy(_getSharedControllerStorage().proxy).doCall(
+ IALMProxy(_getSharedControllerStorage().proxy).doStaticCall(
      oft,
      abi.encodeCall(ILayerZeroLike.quoteSend, (sendParams, false))
  )
```

LOW-05	**LayerZero transfer lacks independent min-receive validation**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Lines: LayerZeroFacet.sol#L197-L206

`LayerZeroFacet.transfer()` builds a `SendParam` with `minAmountLD: 0`, calls `quoteOFT()` to obtain the current expected receive amount, and then sets `minAmountLD` to the quoted `amountReceivedLD` before executing `send()`. Because `minAmountLD` is derived entirely from the OFT's own quote, the slippage check is circular. If the OFT applies unexpected fees or its quoting behaviour changes (e.g., through an upgrade), the transfer would silently accept a reduced amount on the destination chain rather than reverting.

Recommendation

We recommend removing the `quoteOFT()` call and computing `minAmountLD` locally by removing dust: `(amount / decimalConversionRate) * decimalConversionRate`. This mirrors `OFTCoreUpgradeable._removeDust()` logic and avoids trusting an external quote. This causes unexpected fee-bearing OFTs to revert rather than silently accepting losses.

LOW-06	**FarmFacet.withdraw() blocked by failing getReward()**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Line: FarmFacet.sol#L70

`FarmFacet.withdraw()` makes two sequential calls via `ALMProxy`: `IFarmLike.withdraw(amount)` followed unconditionally by `IFarmLike.getReward()`. Because `ALMProxy.doCall()` propagates reverts, any failure in `IFarmLike.getReward()` reverts the entire transaction including the withdrawal. The StakingRewards contract's own comments acknowledges that `getReward()` can fail if the farm is underfunded:

```
/// Before calling this function, the caller should send at least `reward` tokens to the
contract.
/// Otherwise, if the amount sent is less than `reward` passed as a parameter,
/// the unclaimed rewards of other users would be used in the new period.
/// This would result in missing tokens, which might cause `getReward` to fail for some
users.
/// We advise the use of wrapper contracts to perform the transfer and call this function
atomically.

function notifyRewardAmount(uint256 reward) ...
```

Recommendation

We recommend wrapping the `IFarmLike.getReward()` call in a `try/catch`, allowing withdrawal to succeed regardless of reward claim availability.

LOW-07	Curve withdrawal path can be blocked by oracle calls	Acknowledged

Description

Lines:

- CurveFacet.sol#L238
- CurveStableSwapNG.vy#L447

The withdrawal path in `CurveFacet.removeLiquidity()` calls `ICurvePoolLike(pool).stored_rates()` and `ICurvePoolLike(pool).get_virtual_price()` before executing the actual Curve pool withdrawal. The facet uses these values for slippage validation and aggregated rate limit computation. Both `stored_rates()` and `get_virtual_price()` internally call `_stored_rates()` in Curve's stableswap-ng implementation, which performs external calls to oracles for oracle-rate tokens. Curve's proportional `remove_liquidity()` does not depend on these oracle calls, as it computes withdrawn amounts purely from stored balances and succeeds regardless of oracle state.

If any of these external oracle calls revert (e.g., due to oracle downtime), the entire withdrawal transaction reverts, even though the underlying Curve withdrawal would have succeeded. Additionally, if a rate provider is manipulated to return an inflated rate, the computed `valueWithdrawn` becomes artificially large, which may cause a rate limit exceeded revert, again blocking the withdrawal. While `DEFAULT_ADMIN_ROLE` can call `RateLimits.setUnlimitedRateLimitData()` to bypass rate limits, this requires a governance spell to be written, reviewed, and executed, which is too slow for emergency scenarios.

This is particularly concerning because oracle downtime or manipulation is exactly the scenario where the protocol most urgently needs to withdraw funds from the affected pool.

Recommendation

We recommend introducing a dedicated emergency withdrawal function callable by a fast-response emergency role that does not rely on `stored_rates()` or `get_virtual_price()` and bypasses rate limiting.

Customer's Response

Oracles in Curve are trusted.

LOW-08	**Donation attack on ERC4626 vault withdrawal**	Acknowledged

Description

Lines: ERC4626Facet.sol#L157-L158

In `ERC4626Facet.withdraw()`, the rate limit is decreased by `assets`, which is derived from the ALM proxy's balance difference rather than the requested `amount`. When the vault processes the withdrawal through `IERC4626Like.withdraw()`, it may interact with underlying strategies to source liquidity. A compromised or malicious strategy could donate extra assets directly to the ALM proxy during this call, inflating the `assets` value beyond the requested `amount`. A compromised strategy could exploit this to intentionally block withdrawals by donating extra assets to the ALM proxy during the call, inflating `assets` enough to exceed the withdrawal rate limit.

Same issue applies to `AaveFacet.withdraw()`, `BasinFacet.withdraw()`, `PSM3Facet.withdraw()`, `FarmFacet.withdraw()`, `PendleFacet.redeem()`, and `UniswapV3Facet.swap()`, which all use a balance-derived value instead of the requested amount for rate limiting.

Recommendation

We recommend decreasing the rate limit by the requested `amount` instead of the balance-derived `assets`, and adding a validation that the proxy received at least the requested amount:

```
function withdraw(address token, uint256 amount, uint256 maxSharesIn) ... {
  ...

  require(shares <= maxSharesIn, "ERC4626Facet/shares-burned-too-high");
+ require(assets >= amount,      "ERC4626Facet/assets-received-too-low");

- _decreaseRateLimit(getWithdrawRateLimitKey(token),          assets);
- _tryIncreaseRateLimit(getDepositRateLimitKey(token, asset), assets);
+ _decreaseRateLimit(getWithdrawRateLimitKey(token),          amount);
+ _tryIncreaseRateLimit(getDepositRateLimitKey(token, asset), amount);
}
```

Customer's Response

For some facets, there is no amount passed in by the allocator that can be used to affect the rate limit in withdrawals, so we can only rely on either the return of the function call (which we prefer not to do) or the balance delta. For other facets, we don't want a situation where a compromised allocator colludes with a compromised protocol to decrease a rate limit by less than the actual delta, or increase a rate limit by more than the actual delta, if we just rely on the value of the allocator passes in.

INFORMATIONAL-01	**Comment inconsistencies across the codebase**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Lines:

- IUniswapV3Facet.sol#L40 – NatSpec says `tokenId` is "0 for new mints," but the event emits the resulting NFT ID (never zero). The comment describes the input convention, not the emitted value.
- UniswapV3Facet.sol#L168 – `MIN_TICK` / `MAX_TICK` cite Uniswap V4 `TickMath.sol` as their source in a V3 facet. The values are identical across V3 and V4, so this is not functionally incorrect, but the source reference is misleading.
- ICentrifugeFacet.sol#L8-L9 – The NatSpec header says the facet "supports deposit/redeem request lifecycle (submit, cancel, claim)," but the facet only supports cancel, claim-cancel, and transfer functions. Deposit/redeem requests are submitted via `ERC7540Facet`.
- OTCFacet.sol#L237 – Comment says "maxSlippages" (plural) while the variable is `maxSlippage` (singular).

Recommendation

We recommend fixing all mentioned comments.

INFORMATIONAL-02	**Documentation inconsistencies across the codebase**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Lines:

- SECURITY.md#L81 states `doCallWithValue` is gated by `DEFAULT_ADMIN_ROLE` (governance), but it is actually gated by `CONTROLLER` role.
- ARCHITECTURE.md#L24 states "Reentrancy protection across all facet calls," implying Controller-level enforcement. In practice, the Controller's `fallback()` does not use `nonReentrant`, the protection is defined in individual facets.
- ARCHITECTURE.md#L96 states "All contracts in this repo inherit and implement the `AccessControl` contract from OpenZeppelin." which isn't true for the new version.
- LIQUIDITY_OPERATIONS.md#L97 – Uniswap V4 slippage section states "The slippage check uses the pool's virtual price to ensure minimum acceptable returns." which isn't true and seems to be copy-pasted from the Curve section.
- RATE_LIMITS.md#L16 states "Preparing a USDe burn requires the rate limit key `LIMIT_SUSDE_COOLDOWN` to be set" which isn't true.
- CODE_NOTES.md#L153 – `makeAddressAddressKey` helper table only lists `TransferAssetFacet`. Both BasinFacet.sol#L114 and UniswapV3Facet.sol#L909 also use this helper.

Recommendation

We recommend fixing all mentioned documentation inconsistencies.

INFORMATIONAL-03	**OTCBuffer and WEETHModule use legacy storage namespace**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Lines:

- OTCBuffer.sol#L34
- WEETHModule.sol#L67

`OTCBuffer` and `WEETHModule` derive their ERC-7201 storage slots from the legacy `almController.storage.*` namespace, while every other contract in the codebase uses `sky.pau.storage.*`.

Impact: No runtime collision occurs today since each namespace produces a unique slot. However, changing the namespace later requires a storage migration for any deployed proxies, as the derived slot changes with the string.

Recommendation

We recommend aligning both contracts with the rest of the codebase.

INFORMATIONAL-04	**Behavioral change in `UniswapV4Facet._approveWithPermit2` between v1.11 and v1.12**	Fixed - documented this behavioral change in the v1.12 release notes.

Description

Lines:

- UniswapV4Facet.sol#L376-L390
- UniswapV4Lib.sol#L281-L320

In v1.11, `UniswapV4Lib._approveWithPermit2()` unconditionally resets the ERC20 allowance to zero via a low-level call (ignoring failures) before setting the new allowance. In v1.12, this is replaced by `ApproveLib.approve()`, which uses an optimistic approach: it attempts the approval directly and only resets to zero and retries if the initial call fails. Unlike v1.11, the v1.12 fallback path uses reverting calls (`IALMProxy.doCall`), so a token whose `approve(spender, 0)` reverts would cause the transaction to fail, whereas v1.11 silently ignored such failures.

Impact: The change is functionally equivalent for standard ERC20 tokens, but may revert for tokens where `approve(spender, 0)` itself reverts. This difference should be noted in the release notes.

Recommendation

We recommend documenting this behavioral change in the v1.12 release notes.

INFORMATIONAL-05	**`ReentrancyGuard` inheritance is redundant in `AccessControls`**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Line: AccessControls.sol#L15

`AccessControls` inherits `ReentrancyGuard`, but none of its functions use the `nonReentrant` modifier.

Recommendation

We recommend removing the `ReentrancyGuard` inheritance.

INFORMATIONAL-06	**Inaccurate NatSpec on excess native fee refund in `LayerZeroFacet.transfer()`**	Fixed at a84fe9616412e3b6d10ea64301a3481f7dcf9a05

Description

Line: ILayerZeroFacet.sol#L82

The `ILayerZeroFacet.transfer()` NatSpec states "Excess native fee is refunded to the caller," but the implementation passes `proxy` as the `refundAddress` to LayerZero `send()` and then calls `Facet._sweepETH()`, which forwards the entire controller ETH balance to the ALM proxy. The caller never receives a refund.

Recommendation

We recommend updating the NatSpec to reflect actual behavior:

```
-     * @notice Excess native fee is refunded to the caller.
+     * @notice Excess native fee is swept to the ALM proxy.
```

INFORMATIONAL-07	**CentrifugeFacet has an implicit dependency on ERC7540Facet**	Acknowledged

Description

`CentrifugeFacet` provides cancel and claim-cancel operations for deposits and redeems, but the corresponding `ERC7540Facet.requestDeposit()` and `ERC7540Facet.requestRedeem()` that initiate these requests are in `ERC7540Facet`. This creates an implicit dependency, `CentrifugeFacet` can only function when `ERC7540Facet` is also present in the diamond, which is not enforced on smart contract level.

Recommendation

We recommend either making `CentrifugeFacet` self-contained with its own deposit/redeem request functionality and Centrifuge-specific rate limit keys, or explicitly documenting and enforcing the dependency on `ERC7540Facet` at deployment time.

INFORMATIONAL-08	**Missing `msg.value` validation in `LayerZeroFacet.transfer` produces unclear revert**	Acknowledged

Description

Line: LayerZeroFacet.sol#L216

In `LayerZeroFacet.transfer()`, if `msg.value < fee.nativeFee`, the transaction reverts inside the low-level `doCallWithValue{value: fee.nativeFee}` call due to insufficient Controller balance, rather than with a descriptive facet error.

Recommendation

Add an explicit check before the external call for a clear revert message:

```
+       require(msg.value >= fee.nativeFee, "LayerZeroFacet/insufficient-value");
        IALMProxy(proxy).doCallWithValue{value: fee.nativeFee}(
            oftAddress,
            abi.encodeCall(ILayerZeroLike.send, (sendParams, fee, proxy)),
            fee.nativeFee
        );
```

Customer's Response

Internal error is emitted, no need for facet specific error for everything.

INFORMATIONAL-09	`AccessControls.setRoleRevoker()` allows setting a revoker for `DEFAULT_ADMIN_ROLE`	Acknowledged

Description

Line: AccessControls.sol#L70

`AccessControls.setRoleRevoker()` does not prevent setting a revoker for `DEFAULT_ADMIN_ROLE`. A misconfiguration may allow a lower-privileged role to revoke `DEFAULT_ADMIN_ROLE`, potentially removing all administrators.

Recommendation

We recommend adding a check to prevent configuring a revoker for `DEFAULT_ADMIN_ROLE`:

```
  function setRoleRevoker(bytes32 role, bytes32 revokerRole)
      public
      override
      onlyRole(DEFAULT_ADMIN_ROLE)
  {
+     require(role != DEFAULT_ADMIN_ROLE, CannotSetRevokerForAdmin());
      _setRoleAdmin(role, revokerRole);
  }
```

Customer's Response

We're moving to use AccessControls without overriding and then putting more bespoke logic into a module.


UNJVARIANT