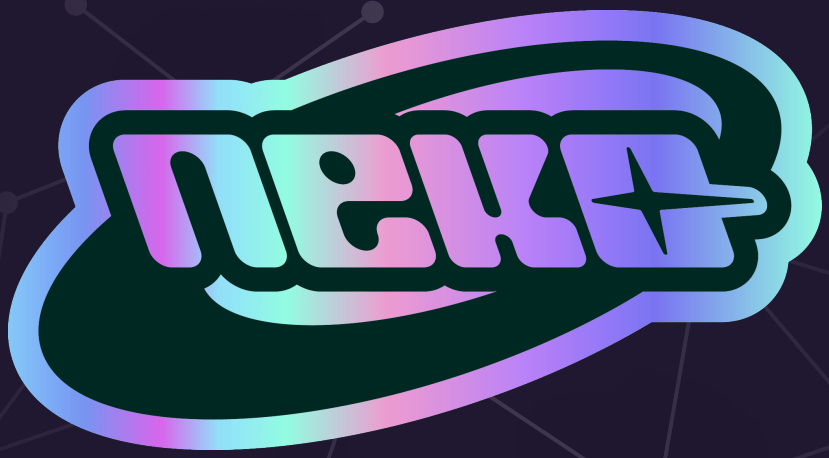
SB SECURITY

Neko.hl

Security Review



Sep 30, 2025

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

Overview

Project	Neko.hl
Repository	Private
Commit Hash	7a0a2e1f356c2491bea7ba8be35e264cf0871bec
Resolution	d1712f82f612648f31a5857eb7a4aff862f42b99
Timeline	September 23 - September 25, 2025

Scope

UniversalEscrowAdapter.sol
StrategyEscrow.sol
UniversalValuerOffchain.sol

Issues Found

Critical Risk	1
High Risk	1
Medium Risk	7
Low/Info Risk	18

5. Findings

5.1. Critical severity

5.1.1. `Valuer::getTotalValue` won't indicate in case one of the strategies have stale values

Severity: Critical Risk

Description: `getTotalValue` will skip including the strategy's value in case it's stale. This is extremely dangerous as it results in a sharp value drop leading to a cheaper share price in the vault, allowing malicious users to deposit when this happens and exit when a new update for this strategy is pushed, raising the share price again.

```
function getTotalValue(address escrow) external view override returns (uint256 totalValue) {
      // This would aggregate all strategy values for the escrow
      // In practice, would need strategy enumeration logic
      // For now, simplified implementation

      bytes32[] memory strategies = _getActiveStrategies(escrow);

      for (uint256 i = 0; i < strategies.length; i++) {
          ValueReport memory report = latestReports[strategies[i]];

          // Skip stale or low confidence values
          if (block.timestamp <= report.timestamp + MAX_STALENESS &&
              report.confidence >= defaultConfidenceThreshold) {
              totalValue += report.value;
          }
      }

      // Add idle assets
      totalValue += IERC20(asset).balanceOf(escrow);

      return totalValue;
}
```

Recommendation: Ensure you always have a proper valuation of the real assets. `fallbackValue` is an option, but since it doesn't require frequent updates, there's a huge probability that both values are stale at the same time. Despite the fix you've decided to go with, `getTotalValue` must not revert as per the specs of the Vault.

Resolution: Fixed

5.2. High severity

5.2.1. Escrow and Adapter have a circular dependency

Severity: High Risk

Description: StrategyEscrow and UniversalEscrowAdapter have the other one as a variable. However, both variables are immutable and are set in the constructor, which prevents one of them from being deployed before the other.

```
contract UniversalEscrowAdapter is IUniversalEscrowAdapter {
    using SafeERC20Lib for IERC20;

    /* IMMUTABLES */

    address public immutable parentVault;
    address public immutable escrow; <----------------------
```

```
contract StrategyEscrow is IStrategyEscrow {
    using SafeERC20Lib for IERC20;

    /* IMMUTABLES */

    address public immutable adapter; <-----------------------------------
    address public immutable owner;
```

Recommendation: Make them with setters.

Resolution: Fixed

5.3. Medium severity

5.3.1. tokens != vault's asset are locked in the UniversalEscrow in case of an emergency

Severity: Medium Risk

Description: In case of an emergency, the owner will be able to restore all funds from the escrow account in the adapter and send the asset to the vault so that users can withdraw their share.

But since the escrow is entering strategies and from the strategies will accumulate: Points, Reward tokens, etc. All these tokens will not be forwarded to the vault, which is true, but will be locked in the adapter.

Recommendation: Add a skim function in the adapter that allows withdrawal of all tokens, not `vault.asset`.

Resolution: Fixed

5.3.2. Deallocation is capping to deposited amount, excluding the yield

Severity: Medium Risk

Description: `deallocate` is capping the amount that can be withdrawn to the amount deposited. This is wrong, because `realAssets` indicate there's a profit from the vault, accumulated by the strategies, but actually it cannot be withdrawn, due to the restriction:

```
function deallocate(
      bytes memory data,
      uint256 assets,
      bytes4,
      address
  ) external override onlyVault returns (bytes32[] memory ids, int256 change) {
      if (data.length == 0) revert InvalidData();

      (bytes32 strategyId, uint256 amount, bytes memory params) = abi.decode(
          data,
          (bytes32, uint256, bytes)
      );

      uint256 actualAmount = amount > assets ? assets : amount;
->      uint256 currentAllocation = allocations[strategyId];
->      actualAmount = actualAmount > currentAllocation ? currentAllocation : actualAmount;

      if (actualAmount > 0) {
          // Execute strategy-specific deallocation in escrow
          IStrategyEscrow.Call[] memory calls = _buildDeallocationCalls(
              strategyId,
              actualAmount,
              params
          );

          if (calls.length > 0) {
              IStrategyEscrow(escrow).executeMulticall(strategyId, calls);
          }

          allocations[strategyId] -= actualAmount;

          // Remove from active strategies if fully deallocated
          if (allocations[strategyId] == 0) {
              _removeActiveStrategy(strategyId);
          }

          emit StrategyDeallocated(strategyId, actualAmount);
      }

      ids = new bytes32[](1);
      ids[0] = strategyId;
      return (ids, -int256(actualAmount));
  }
```

Recommendation: Only after fixing [M-04], consider applying the following modifications.

```
function deallocate(
      bytes memory data,
      uint256 assets,
      bytes4,
      address
  ) external override onlyVault returns (bytes32[] memory ids, int256 change) {
      if (data.length == 0) revert InvalidData();

      (bytes32 strategyId, uint256 amount, bytes memory params) = abi.decode(
          data,
          (bytes32, uint256, bytes)
      );

      uint256 actualAmount = amount > assets ? assets : amount;
-     uint256 currentAllocation = allocations[strategyId];
+     uint256 currentAllocation = valuer.getValue(strategyId);
      actualAmount = actualAmount > currentAllocation ? currentAllocation : actualAmount;

      if (actualAmount > 0) {
          // Execute strategy-specific deallocation in escrow
          IStrategyEscrow.Call[] memory calls = _buildDeallocationCalls(
              strategyId,
              actualAmount,
              params
          );

          if (calls.length > 0) {
              IStrategyEscrow(escrow).executeMulticall(strategyId, calls);
          }

-         allocations[strategyId] -= actualAmount;
+         allocations[strategyId] -= actualAmount > allocations[strategyId] ? allocations[strategyId] :
actualAmount

          // Remove from active strategies if fully deallocated
          if (allocations[strategyId] == 0) {
              _removeActiveStrategy(strategyId);
          }

          emit StrategyDeallocated(strategyId, actualAmount);
      }

      ids = new bytes32[](1);
      ids[0] = strategyId;
      return (ids, -int256(actualAmount));
  }
```

Additionally, ensure there's a mechanism to manually unwind the needed assets from the underlying strategies.

Resolution: Fixed

5.3.3. Weird ERC20s don't work

Severity: Medium Risk

Description: Weird ERC20s don't work in the Neko Vault, because the VaultV2 contract itself can't do proper share accounting if there are fees on transfer or dynamic balance changes. Moreover, since the Vault ↔ Adapter transfers happen based on the arguments provided to `allocate` and `deallocate` functions, rather than the tokens actually received, it will mess up the `allocations` mapping, resulting in mismatch between what entered the strategies, vs. what was written in the storage:

```
function allocate(
    bytes memory data,
    uint256 assets,
    bytes4,
    address
) external override onlyVault notEmergency returns (bytes32[] memory ids, int256 change) {
    if (data.length == 0) revert InvalidData();

    (bytes32 strategyId, uint256 amount, bytes memory params) = abi.decode(
        data,
        (bytes32, uint256, bytes)
    );

    if (strategyPaused[strategyId]) revert StrategyPaused();

    // Transfer assets to escrow
    if (amount > 0 && amount <= assets) {
        SafeERC20Lib.safeTransfer(asset, escrow, amount);
        allocations[strategyId] += amount;

        // Notify escrow of allocation
        IStrategyEscrow(escrow).notifyAllocation(strategyId, amount);

        // Track active strategy
        _addActiveStrategy(strategyId);

        emit StrategyAllocated(strategyId, amount);
    }

    // Return allocation info
    ids = new bytes32[](1);
    ids[0] = strategyId;
    return (ids, int256(amount));
}
```

Recommendation:

1. Avoid using ERC20 tokens that deviate from the standard.

2. In `allocate`, in the `if` check use the `assets` arguments, because it shows the actually forwarded tokens, whereas amount from data can be inaccurate.

Resolution: Fixed

5.3.4. `VaultV2::exit` will revert due to multiple occasions

Severity: Medium Risk

Description: ONLY If the `UniversalEscrowAdapter` is enabled as `liquidityAdapter` in the Vault: instant exits will be impossible, due to multiple occasions:

1. when the lowest between `assets/amount/allocations[strategyId]` is lower than the required amount - `assets`.

2. since the `liquidityData` is storage variable, given by the allocator, it won't be matching the assets.

- `amount >= assets` - works

- `assets > amount` - will revert as there's insufficient tokens withdrawn from the strategies.

```
function deallocate(
    bytes memory data,
    uint256 assets,
    bytes4,
    address
) external override onlyVault returns (bytes32[] memory ids, int256 change) {
    if (data.length == 0) revert InvalidData();

    (bytes32 strategyId, uint256 amount, bytes memory params) = abi.decode(
        data,
        (bytes32, uint256, bytes)
    );

    uint256 actualAmount = amount > assets ? assets : amount;
    uint256 currentAllocation = allocations[strategyId];
    actualAmount = actualAmount > currentAllocation ? currentAllocation : actualAmount;

    if (actualAmount > 0) {
        // Execute strategy-specific deallocation in escrow
        IStrategyEscrow.Call[] memory calls = _buildDeallocationCalls(
            strategyId,
            actualAmount,
            params
        );

        if (calls.length > 0) {
            IStrategyEscrow(escrow).executeMulticall(strategyId, calls);
        }

        allocations[strategyId] -= actualAmount;

        // Remove from active strategies if fully deallocated
        if (allocations[strategyId] == 0) {
            _removeActiveStrategy(strategyId);
        }

        emit StrategyDeallocated(strategyId, actualAmount);
    }

    ids = new bytes32[](1);
    ids[0] = strategyId;
    return (ids, -int256(actualAmount));
}
```

Recommendation: Don't use the amount, even consider removing it from the struct, as the data use case is to give any information, excluding the amount that has to be withdrawn. Either withdraw exact `assets` or revert the execution - this will simplify the logic.

Resolution: Fixed

5.3.5. update functions in Values must exclude pending but not deactivated signers

Severity: Medium Risk

Description: `UniversalValuerOffchain.updateValue()` works with signatures and needs to pass a `totalWeight` to continue the transaction. `_verifySignatures()` is used for this purpose and sums up the weights of all signatures, but since the signer can be deactivated and deactivation is a two-step process, `_verifySignatures()` forgets to check if the signer has a pending deactivation.

```
function _verifySignatures(
    bytes32 strategyId,
    uint256 value,
    uint256 confidence,
    uint256 nonce,
    uint256 expiry,
    bytes[] calldata signatures
) internal view returns (uint256 totalWeight) {
    MORE CODE

    // Track used signers to prevent duplicates
    address[] memory usedSigners = new address[](signatures.length);
    uint256 usedCount = 0;

    for (uint256 i = 0; i < signatures.length; i++) {
        address signer = _recoverSigner(ethSignedHash, signatures[i]);

        // Skip if signer already counted
        bool alreadyUsed = false;
        for (uint256 j = 0; j < usedCount; j++) {
            if (usedSigners[j] == signer) {
                alreadyUsed = true;
                break;
            }
        }

        if (alreadyUsed) continue;

        if (signers[signer].authorized) { <--------------------------------------
            totalWeight += signers[signer].weight;
            usedSigners[usedCount] = signer;
            usedCount++;
        }
    }

    return totalWeight;
}
```

Recommendation: Inside `_verifySignatures` check if `if (signers[signer].authorized && (!pendingSignerRemoval[signer] || signerChangeTimestamp[signer] > block.timestamp)`, if yes - include.

Resolution: Fixed

5.3.6. confidence check is missing in update functions of the Valuer

Severity: Medium Risk

Description: Each new value report added via `updateValue()` has confidence, but it is not checked whether the confidence is ≥ `minConfidence` of the strategy.

```
function updateValue(
    bytes32 strategyId,
    uint256 value,
    uint256 confidence,
    uint256 nonce,
    uint256 expiry,
    bytes[] calldata signatures
) external override notEmergency {
    ValueReport memory lastReport = latestReports[strategyId];

    // Validate nonce to prevent replay
    if (nonce <= lastReport.nonce) revert StaleNonce();

    // Validate signature expiry
    if (expiry < block.timestamp) revert SignatureExpired();
    if (expiry > block.timestamp + MAX_SIGNATURE_AGE) revert SignatureExpiryTooFar();

    // Check minimum update interval (unless significant change)
    UpdateConfig memory config = updateConfigs[strategyId];
    uint256 changePercent = _calculateChangePercent(lastReport.value, value);

    if (block.timestamp < lastReport.timestamp + config.minUpdateInterval) {
        // Only allow update if change exceeds threshold
        if (changePercent < config.pushThreshold) {
            revert UpdateTooFrequent();
        }
    }

    // Validate price bounds
    _validatePriceBounds(strategyId, lastReport.value, value);

    // Verify signatures with duplicate prevention
    ....

    if (totalWeight < requiredWeight) revert InsufficientSignatures();

    // Store the new report
    latestReports[strategyId] = ValueReport({
        value: value,
        timestamp: block.timestamp,
        confidence: confidence, <---------------------------------
        nonce: nonce,
        isPush: true,
        lastUpdater: msg.sender
    });

    emit ValueUpdated(strategyId, value, confidence, block.timestamp, true);
}
```

Recommendation: Add a check for `minConfidence`.

Resolution: Fixed

5.3.7. Strategy can be in a state where it cannot be changed in `UniversalValuerOffchain` smart contract

Severity: Medium Risk

Description: Particular strategy can be in a state where it cannot be changed via `UniversalValuerOffchain.sol#updateValue()` function even if `changePercent` is greater than `config.pushThreshold`. A push update passes the `pushThreshold` check but then reverts in `_validatePriceBounds` if the same change exceeds `maxPriceChangeBps` (meanwhile `batchUpdateValues()` never checks bounds at all - inconsistent behavior). That leaves a strategy in a stuck state: the valuer/allocators expect a large change to be push-able, but the bounds check blocks it.

```
function updateValue(
    bytes32 strategyId,
    uint256 value,
    uint256 confidence,
    uint256 nonce,
    uint256 expiry,
    bytes[] calldata signatures
) external override notEmergency {

    // ... code ...

    uint256 changePercent = _calculateChangePercent(lastReport.value, value);

    if (block.timestamp < lastReport.timestamp + config.minUpdateInterval) {
        // Only allow update if change exceeds threshold
        if (changePercent < config.pushThreshold) {
            revert UpdateTooFrequent();
        }
    }

    // Validate price bounds
    _validatePriceBounds(strategyId, lastReport.value, value);

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

Example

- `lastValue = 100`

- `newValue = 140` → `changePercent = (40*10000)/100 = 4000 bps = 40%`

- `config.pushThreshold = 3000` (30%) -> passes pushThreshold

- `maxPriceChangeBps[strategyId] = 2000` (20%) -> fails bounds -> `updateValue()` will revert at `_validatePriceBounds`.

Recommendation: Require `pushThreshold <= maxPriceChangeBps[strategyId]` inside `configureStrategy()` (or when setting `maxPriceChangeBps`).

Resolution: Fixed

5.1. Low severity

5.1.1. Useless double calculation of `changePercent` in `UniversalValuerOffchain.updateValue()`

Severity: Low Risk

Description: `updateValue()` checks if the new value is at least `pushThreshold` % greater than the last stored value for a given `strategyId`. However, since there is a min(`pushThreshold`) and a `MAX_PRICE_CHANGE_BPS`, `updateValue()` calculates `changePercent` twice.

```
uint256 changePercent = _calculateChangePercent(lastReport.value, value);  <----------------------

if (block.timestamp < lastReport.timestamp + config.minUpdateInterval) {
    // Only allow update if change exceeds threshold
    if (changePercent < config.pushThreshold) {
        revert UpdateTooFrequent();
    }
}

// Validate price bounds
_validatePriceBounds(strategyId, lastReport.value, value);
```

```
function _validatePriceBounds(bytes32 strategyId, uint256 oldValue, uint256 newValue) internal view {
    if (oldValue == 0) return; // No bounds check for initial value

    uint256 maxChange = maxPriceChangeBps[strategyId];
    if (maxChange == 0) {
        maxChange = MAX_PRICE_CHANGE_BPS; // Use default if not set
    }

    uint256 changePercent = _calculateChangePercent(oldValue, newValue);
    if (changePercent > maxChange) {
        revert PriceChangeExceedsBounds(changePercent, maxChange);
    }
}
```

Recommendation: Use the calculated one in `_validatePriceBounds`.

Resolution: Fixed

5.1.2. `batchUpdateValues` is missing constraints for `changePercent` and `minUpdateInterval`

Severity: Low Risk

Description: `batchUpdateValues` should be the same as `updateValue`, but in a loop. Although there are no checks for `changePercent` or `minUpdateInterval`.

```
function batchUpdateValues(
    bytes32[] calldata strategyIds,
    uint256[] calldata values,
    uint256[] calldata confidences,
    uint256 nonce,
    uint256 expiry,
    bytes[] calldata signatures
) external override notEmergency {
    if (strategyIds.length != values.length ||
        strategyIds.length != confidences.length) {
        revert ArrayLengthMismatch();
    }

    // Validate signature expiry
    if (expiry < block.timestamp) revert SignatureExpired();
    if (expiry > block.timestamp + MAX_SIGNATURE_AGE) revert SignatureExpiryTooFar();

    // Verify signatures for batch
    bytes32 batchHash = keccak256(abi.encode(strategyIds, values, confidences, nonce, expiry));
    uint256 totalWeight = _verifyBatchSignatures(batchHash, signatures);

    if (totalWeight < requiredWeight) revert InsufficientSignatures();

    // Update all values
    for (uint256 i = 0; i < strategyIds.length; i++) {
        bytes32 strategyId = strategyIds[i];

        // Check nonce for each strategy
        if (nonce <= latestReports[strategyId].nonce) continue;

        latestReports[strategyId] = ValueReport({
            value: values[i],
            timestamp: block.timestamp,
            confidence: confidences[i],
            nonce: nonce,
            isPush: true,
            lastUpdater: msg.sender
        });

        emit ValueUpdated(strategyId, values[i], confidences[i], block.timestamp, true);
    }
}
```

Recommendation: Make sure `batchUpdateValues()` to match all checks from `updateValue()`.

Resolution: Fixed

5.1.3. Missing `defaultConfidenceThreshold` setter

Severity: Low Risk

Description: In UniversalValuerOffchain.sol, `defaultConfidenceThreshold` is preset to 95% but is neither immutable nor constant, thus it's logical to have a setter function, which is currently missing.

Recommendation: Add setter for `defaultConfidenceThreshold`.

Resolution: Fixed

5.1.4. Values for `trackToken/isTracked` in `StrategyEscrow` smart contract cannot be removed

Severity: Low Risk

Description: `StrategyEscrow#trackToken(address)` lets the owner add a token to `trackedTokens` and sets `isTracked[token] = true`, but there is no way to remove a token or set `isTracked[token] = false`. The tracked list can therefore grow with stale entries and cannot be cleaned up. This causes unnecessary gas cost when iterating `trackedTokens`, prevents proper housekeeping (or re-use of a token slot) and makes it impossible to de-register a token (or to ensure `emergencyWithdrawAll`/other logic only touches currently relevant tokens).

Recommendation: Add an owner-only `untrackToken(address token)` that clears `isTracked[token]` and removes the token from `trackedTokens` (swap-and-pop).

Resolution: Fixed

5.1.5. Enforce constraints in `UniversalValuerOffchain::configureStrategy()` function

Severity: Low Risk

Description: `UniversalValuerOffchain.sol#configureStrategy()` function currently accepts arbitrary values. That allows owners to set nonsensical or dangerous parameters (e.g. `minUpdateInterval` below protocol minimum, `maxStaleness` above safe limit, `pushThreshold` larger than allowed price-change bounds, or `minConfidence` weaker than the system default). These misconfigurations can break update logic.

Recommendation: Validate inputs in `configureStrategy()` and revert on out-of-range values. Enforce:

- minUpdateInterval >= MIN_UPDATE_INTERVAL
- maxStaleness <= MAX_STALENESS
- pushThreshold <= MAX_PRICE_CHANGE_BPS
- minConfidence >= defaultConfidenceThreshold

Resolution: Fixed

5.1.6. Don't allow the `UniversalValuerOffchain::requestUpdate()` function to be called during emergency mode

Severity: Low Risk

Description: During emergency mode the valuer should not accept normal update requests. `requestUpdate()` currently can be called while `emergencyMode == true`, which can trigger on-chain signals, emit misleading `UpdateRequested` events and encourage off-chain actors to push updates when the system is in an emergency state.

Recommendation: Block `requestUpdate()` when emergencyMode is enabled. Add the existing `notEmergency` modifier to `requestUpdate()` so it reverts instead of emitting events during emergency.

Resolution: Fixed

5.1.7. Recovery timelock in Adapter not freezing operations - `allocate`/`deallocate` allowed during pending emergency

Severity: Low Risk

Description: `initiateEmergencyRecovery()` sets `emergencyRecoveryPending` and a timelock timestamp but **does not** set `emergencyMode`. Until `forceRecovery()` runs after the timelock, `emergencyMode` remains false and `notEmergency()` checks still pass. That means `allocate()`, `deallocate()` and `StrategyEscrow.executeMulticall()` can be executed while a recovery is pending. During the 24h window an allocator or agent can move or reassign funds, changing the state that the owner expected to freeze for recovery.

Example: owner initiates recovery → allocator or agent deallocates or shuffles assets during the 24h → `forceRecovery()` recovers less than expected or leaves inconsistent state.

Recommendation: Block state-changing operations during the recovery timelock.

Resolution: Fixed

5.1.8. Use config's `maxStaleness` and `minConfidence` in `getValue()` and `getTotalValue()` of the Valuer

Severity: Low Risk

Description: `getValue()` and `getTotalValue()` use hard-coded/outer variables (`MAX_STALENESS` and `defaultConfidenceThreshold`) instead of the strategy-specific `config` values. Because `config.maxStaleness` and `config.minConfidence` are updatable, the current code can accept or reject reports contrary to the configured policy - stale/low-confidence reports may be incorrectly allowed or valid updates incorrectly rejected. Also `defaultConfidenceThreshold` is effectively immutable in the codebase, creating mismatch and future maintenance risk.

Recommendation: Use the per-strategy config values when checking staleness and confidence. Replace `MAX_STALENESS` with `config.maxStaleness` and `defaultConfidenceThreshold` with `config.minConfidence` (loaded from the strategy's current config) in both `getValue()` and the loop inside `getTotalValue()`.

Resolution: Fixed

5.1.9. Rename `onlyStrategy` to `onlyStrategyAgentOrOwner` in Adapter

Severity: Low Risk

Description: The `onlyAgent(bytes32)` modifier allows either the `strategyAgents[strategyId]` **or** the `adapter` (when set) to call protected functions, but its name `onlyAgent` is misleading. This obscures the true access control and can confuse reviewers, integrators, and auditors about who is authorized.

Recommendation: Rename the modifier to `onlyAgentOrAdapter` (and update its NatSpec/comment) to reflect actual semantics. Replace all uses and update tests/docs accordingly. Keep the logic unchanged — this is purely a clarity/maintainability fix to avoid mistaken assumptions about access control.

Resolution: Fixed

5.1.10. `UniversalEscrowAdapter::allocate()` store the same `strategylds` twice

Severity: Low Risk

Description: `UniversalEscrowAdapter.allocate()` function calls `IStrategyEscrow(escrow).notifyAllocation()` function and then calls its own `_addActiveStrategy(strategyId)`. That causes the same strategy id to be tracked in two places (escrows activeStrategyList and adapters `activeStrategies`), duplicating state and responsibility. This redundancy increases gas, couples two contracts' invariants, and can lead to out-of-sync or inconsistent state if one side updates but the other fails/rolls back.

```
function allocate(
    bytes memory data,
    uint256 assets,
    bytes4,
    address
) external override onlyVault notEmergency returns (bytes32[] memory ids, int256 change) {
    // ... code ...

    // Transfer assets to escrow
    if (amount > 0 && amount <= assets) {
        SafeERC20Lib.safeTransfer(asset, escrow, amount);
        allocations[strategyId] += amount;

        // Notify escrow of allocation
        IStrategyEscrow(escrow).notifyAllocation(strategyId, amount); <------------

        // Track active strategy
        _addActiveStrategy(strategyId);

        emit StrategyAllocated(strategyId, amount);
    }

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

```
function notifyAllocation(
    bytes32 strategyId,
    uint256 amount
) external override onlyAdapter {
    strategyAllocations[strategyId] += amount;
    _addActiveStrategy(strategyId);

    emit AllocationNotified(strategyId, amount);
}
```

Recommendation: Keep a single canonical source of truth for "active strategies". Either:

- Remove the adapter-side `_addActiveStrategy` and rely on `StrategyEscrow` to maintain the active list (adapter can query escrow when needed); or
- Keep adapter tracking but do not call `notifyAllocation` to mutate escrow's list — instead only notify amounts and leave list management to the adapter.

Resolution: Fixed

5.1.11. Use ECDSA from Oz for safer signature validation

Severity: Low Risk

Description: Although signature validation is easier, OpenZeppelin contracts' use of ECDSA is preferable as it is proven and battle-tested.

```
function _recoverSigner(bytes32 hash, bytes memory signature) internal pure returns (address) {
    if (signature.length != 65) revert InvalidSignature();

    bytes32 r;
    bytes32 s;
    uint8 v;

    assembly {
        r := mload(add(signature, 0x20))
        s := mload(add(signature, 0x40))
        v := byte(0, mload(add(signature, 0x60)))
    }

    if (v < 27) {
        v += 27;
    }

    if (v != 27 && v != 28) revert InvalidSignature();

    return ecrecover(hash, v, r, s);
}
```

Recommendation: Replace the custom `ecrecover` logic with OpenZeppelin's `ECDSA.recover` (and `toEthSignedMessageHash` when appropriate) to handle malleability and edge cases safely.

Resolution: Fixed

5.1.12. `emergencyWithdrawAll` sum all tokens in one variable

Severity: Low Risk

Description: `emergencyWithdrawAll` emit the amount of tokens recovered from the escrow account, but they can have different decimal numbers. Although they are all summed up in one variable.

Recommendation: If you still want to emit it, emitting an array with the tokens and an array with the amounts would be better.

Resolution: Fixed

5.1.13. Tokens can be transferred but not utilized in a strategy

Severity: Low Risk

Description: allocate will send only the `amount` of `data` to the escrow account, and in case the `amount` < `assets`, the difference will be locked in the adapter and will never be used for yield.

```
if (amount > 0 && amount <= assets) {
    SafeERC20Lib.safeTransfer(asset, escrow, amount);
    allocations[strategyId] += amount;

    // Notify escrow of allocation
    IStrategyEscrow(escrow).notifyAllocation(strategyId, amount);
```

Recommendation: Consider using them in some way in your yield strategies.

Resolution: Fixed

5.1.14. Unsanitized data passed from `forceDeallocate` to `UniversalEscrowAdapter`

Severity: Low Risk

Description: User can pass an arbitrary data to `forceDeallocate`, which can force the adaptors into executing malicious operations, for example withdraw the entire allocation. Impact is still not clearly defined because `_buildDeallocationCalls` isn't implemented:

But whenever this is it will make it possible to harm the vault.

Recommendation:

1. Use the `msg.sig` (3rd) argument of the `deallocate`:

2. Have a sanitization or whitelisting logic, allowing the users to do only a predefined set of actions.

3. Optionally, restrict `forceDeallocate` calls to taking only the tokens, available in the Adapter.

Resolution: Fixed

5.1.15. upon full deallocate of the allocation, escrow will continue have the strategy as active

Severity: Low Risk

Description: When a full deallocation is done, the `strategyId` will be removed from `Adapter.activeStrategies`, but since Escrow also stores them (which is reported in other issue), those in the escrow account are not removed.

```
function deallocate(
    bytes memory data,
    uint256 assets,
    bytes4,
    address
) external override onlyVault returns (bytes32[] memory ids, int256 change) {
    if (data.length == 0) revert InvalidData();

    (bytes32 strategyId, uint256 amount, bytes memory params) = abi.decode(
        data,
        (bytes32, uint256, bytes)
    );

    // ... code ...

    if (actualAmount > 0) {
        // ... code ...

        // Remove from active strategies if fully deallocated
        if (allocations[strategyId] == 0) {
            _removeActiveStrategy(strategyId); <------------
        }

        emit StrategyDeallocated(strategyId, actualAmount);
    }

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

```
function _removeActiveStrategy(bytes32 strategyId) internal {
    uint256 length = activeStrategies.length;
    for (uint256 i = 0; i < length; i++) {
        if (activeStrategies[i] == strategyId) {
            activeStrategies[i] = activeStrategies[length - 1];
            activeStrategies.pop();
            break;
        }
    }
}
```

Recommendation: If the other issue is fixed and `activeStrategies` remains in only one source, this will be fixed.

Resolution: Fixed

5.1.16. dailyLimit and usedToday are wrongly working for native only

Severity: Low Risk

Description: Each multicall in Escrow has properties `dailyLimit` and `usedToday`, which from the documentation and by design should limit the number of `vault.assets` that the escrow will move daily.

But currently it uses native ETH forward with calls and does not store the amount of assets.

```
function _executeCall(Call memory call) internal {
    bytes4 selector = bytes4(call.data);
    WhitelistEntry storage entry = whitelist[call.target][selector];

    // Check whitelist
    if (!entry.allowed) revert NotWhitelisted();

    // Check daily limit if applicable
    if (entry.dailyLimit > 0) {
        // Reset daily counter if needed
        if (block.timestamp > entry.lastReset + DAY) {
            entry.usedToday = 0;
            entry.lastReset = block.timestamp;
        }

        // Check limit
        if (entry.usedToday + call.value > entry.dailyLimit) {
            revert DailyLimitExceeded();
        }

        entry.usedToday += call.value; <--------------
    }

    // Execute call
    (bool success, bytes memory result) = call.target.call{value: call.value}(call.data);
    if (!success) revert CallFailed(call.target, call.data);
}
```

Recommendation: It is better to remove this logic, as fixing it and restricting `vault.asset` will restrict emergency withdrawal and the entire project in case of high user exposure.

Resolution: Fixed

5.1.17. Emergency recovery state mismatch - adapter leaves stale allocations after escrow `emergencyWithdrawAll`

Severity: Low Risk

Description: `UniversalEscrowAdapter.forceRecovery()` calls `StrategyEscrow.emergencyWithdrawAll(...)` which clears the escrow's `activeStrategyList`, but the adapter does **not** clear its own `allocations` mapping or `activeStrategies` entries. Result: the escrow shows no active strategies while the adapter still records non-zero allocations - deterministic, provable state mismatch that breaks invariants (wrong `isStrategyActive()`) and can cause incorrect future allocate/deallocate behavior.

```
function forceRecovery() external override onlyOwner {
    if (!emergencyRecoveryPending) revert EmergencyRecoveryNotInitiated();
    if (block.timestamp < emergencyRecoveryTimestamp) revert EmergencyRecoveryTimelockNotExpired();

    // Clear timelock state
    emergencyRecoveryPending = false;
    emergencyRecoveryTimestamp = 0;

    emergencyMode = true;

    // Emergency withdraw all from escrow - now with validated recipient
    IStrategyEscrow(escrow).emergencyWithdrawAll(address(this));

    uint256 recoveredAmount = IERC20(asset).balanceOf(address(this));

    // ... code ...

    emit EmergencyWithdrawal(address(this), netRecovered);
}
```

Recommendation: When performing emergency withdraw, clear per-strategy state in the escrow and in the adapter. In `StrategyEscrow.emergencyWithdrawAll()` iterate the pre-withdraw `activeStrategyList` and `delete strategyAllocations[id]` and `delete activeStrategies[id]` for each id before `delete activeStrategyList`. And in `UniversalEscrowAdapter.forceRecovery()` clear `allocations[strategyId]` and remove `strategyId` from `activeStrategies` for the same list returned/known by the escrow.

Resolution: Fixed

5.1.18. Use config's values, instead of constants

Severity: Low Risk

Description: Some of the constant values - `MIN_UPDATE_INTERVAL` and `MAX_STALENESS` usages can be inaccurate if the config has different values. Since config has a priority over the constants, due to being able to be modified we must use only its values.

```
uint256 private constant MAX_STALENESS = 24 hours;
uint256 private constant MIN_UPDATE_INTERVAL = 5 minutes;
```

Recommendation: Replace the constants from the values of the config in the `UniversalValuerOffchain`.

Resolution: Fixed