yAudit

September 2026

Prepared for
Blueberry

Audited by
Watermelon
HHK

Curvance Lender Strategy

Smart Contract Security Assessment

Contents

1 Review Summary **2**
1.1 Protocol Overview . . . 2
1.2 Audit Scope . . . 2
1.3 Risk Assessment Framework . . . 2
1.3.1 Severity Classification . . . 3
1.4 Key Findings . . . 3
1.5 Overall Assessment . . . 3

2 Audit Overview **4**
2.1 Project Information . . . 4
2.2 Audit Timeline . . . 4
2.3 Audit Resources . . . 4
2.4 Critical Findings . . . 5
2.5 High Findings . . . 5
2.6 Medium Findings . . . 5
2.7 Low Findings . . . 5
2.7.1 Reward swaps lack slippage protection and can be sandwiched . . . 5
2.7.2 `StrategyFactory.isDeployedStrategy` only recognizes the latest strategy for each vault . . . 6
2.8 Gas Savings Findings . . . 7
2.8.1 Factory `newStrategy` methods perform redundant external calls . . . 7
2.9 Informational Findings . . . 8
2.9.1 `positionValue` accepts unregistered markets while `marketMaxWithdraw` rejects them . . . 8
2.9.2 `CurvanceOptimizer` does not override `availableDepositLimit` . . . 9
2.9.3 `addRewardToken` accepts duplicates, which `removeRewardToken` then fails to remove . . . 10
2.9.4 `CurvanceOptimizer.emergencyWithdraw` can revert when destinations share underlying liquidity . . . 11
2.9.5 `removeMarket` can be blocked by donating dust shares to the optimizer . . . 12
2.9.6 Missing event emission in factory `setAddresses` methods . . . 13

1 Review Summary

1.1 Protocol Overview

Curvance Lender strategy is a set of Yearn V3 tokenized strategies that deploy a single asset into Curvance borrowable markets and lending optimizers on Monad. CurvanceLender targets one destination, identified at construction as either a direct borrowable cToken or a LendingOptimizer vault. CurvanceOptimizer allocates across up to sixteen destinations of either kind according to management-set asset targets. Both strategies sell Merkl reward emissions through Yearn auctions or, optionally, configured Uniswap V3 routes.

1.2 Audit Scope

This audit covers 5 smart contracts totaling approximately 665~ lines of code across 1 day of review.

```
src
├── BlueberryProtocolAddressProvider.sol
├── CurvanceOptimizer.sol
├── CurvanceOptimizerFactory.sol
├── Strategy.sol
└── StrategyFactory.sol
```

1.3 Risk Assessment Framework

1.3.1 Severity Classification

Severity	Description	Potential Impact
Critical	Immediate threat to user funds or protocol integrity	Direct loss of funds, protocol compromise
High	Significant security risk requiring urgent attention	Potential fund loss, major functionality disruption
Medium	Important issue that should be addressed	Limited fund risk, functionality concerns
Low	Minor issue with minimal impact	Best practice violations, minor inefficiencies
Undetermined	Findings whose impact could not be fully assessed within the time constraints of the engagement. These issues may range from low to critical severity, and although their exact consequences remain uncertain, they present a sufficient potential risk to warrant attention and remediation.	Varies based on actual severity
Gas	Findings that can improve the gas efficiency of the contracts.	Increased transaction costs
Informational	Code quality and best practice recommendations	Reduced maintainability and readability

Table 1: severity classification

1.4 Key Findings

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	2
Informational	6

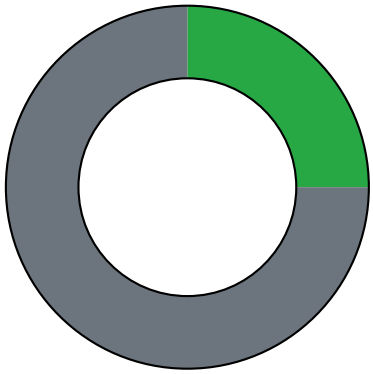

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

The reviewed contracts are well constructed and supported by an unusually thorough test suite, with no critical or high severity issues identified. The most significant finding concerns reward sales, which execute with no minimum output and are therefore sandwichable. The remaining

findings are low severity or informational and cluster around operational robustness rather than fund safety.

2 Audit Overview

2.1 Project Information

Protocol Name: Blueberry
Repository: https://github.com/Blueberryfi/curvance-lender
Commit Hash: b912aa446e2f478a701013cc37a82854851545c2
Commit URL: https://github.com/Blueberryfi/curvance-lender/tree/b912aa446e2f478a701013cc37a82854851545c2

2.2 Audit Timeline

The audit was conducted from September 10 to 11, 2026.

2.3 Audit Resources

- Code repositories and documentation

Category	Mark	Description
Access Control	Good	Roles are cleanly separated across management, keeper and emergency admin, with strategy management transferred through a two-step handover.
Mathematics	Good	There are no bespoke formulas; arithmetic is limited to min and max clamps over ERC-4626 conversions delegated to audited libraries.
Complexity	Average	Both strategies are short and simple, built from small single-purpose functions with a clear split between the single-target and multi-market designs.
Libraries	Good	
Decentralization	Low	Management registers destinations, sets allocation targets and can enable the direct swap route, all effective immediately and without an on-chain timelock.
Documentation	Good	README and the operational documents are thorough and pre-disclose several genuine limitations. NatSpec coverage is uneven between the two strategies.
Testing and verification	Good	Strong coverage, combining pinned-block fork integration with randomized action sequences and targeted regression tests.

Table 2: Code Evaluation Matrix

2.4 Critical Findings

None.

2.5 High Findings

None.

2.6 Medium Findings

None.

2.7 Low Findings

2.7.1 Reward swaps lack slippage protection and can be sandwiched

Technical Details

`CurvanceLender._claimAndSellRewards` swaps the strategy's entire balance of each configured reward token into the underlying asset when auctions are disabled. However, it calls `_swapFrom` with `_minAmountOut` set to zero:

```
_swapFrom(
    rewardToken,
    address(asset),
    ERC20(rewardToken).balanceOf(address(this)),
    0
);
```

`CurvanceLender.manualSellRewards` similarly passes a zero minimum output when keepers sell rewards manually.

The value is forwarded to the Uniswap router as `amountOutMinimum`. Consequently, these swaps will accept any amount of the underlying asset, regardless of the current exchange rate.

An MEV bot can observe a pending reward sale, front-run it to manipulate the pool price, allow the strategy to execute at the unfavorable price, and back-run the transaction to restore the price. Since no minimum output is enforced, the strategy cannot revert when the execution price deviates significantly from the expected price. For sufficiently large reward balances or illiquid pools, an attacker may extract most of the rewards' economic value.

Impact

Low. MEV bots can sandwich direct reward sales and capture up to nearly the entire value of the strategy's accrued rewards. The resulting loss is borne by strategy depositors through reduced reported profits. However the strategy being on Monad and sales being permissioned the likelihood of sandwich is low.

Recommendation

Given that reward tokens are provided in the form of receipt tokens, rather than the direct underlying, it seems unlikely such assets would have an oracle price feed. As a consequence, the most sensible path seems to remove synchronous swaps and favour using auctions instead.

For `manualSellRewards`, a `minAmountOut` calldata parameter may be added, in order to use such value within the `_swapFrom` call.

Developer Response

Agreed. Fixed in PR#6.

2.7.2 `StrategyFactory.isDeployedStrategy` only recognizes the latest strategy for each vault

Technical Details

`StrategyFactory.newStrategy` records each deployed strategy using its underlying vault as the mapping key:

```
deployments[_vault] = address(_newStrategy);
```

If another strategy is subsequently deployed for the same vault, this assignment overwrites the address of the previously deployed strategy. `StrategyFactory.isDeployedStrategy` retrieves the vault associated with the supplied strategy and compares the strategy against the single address currently stored for that vault.

Consequently, the function returns `false` for valid strategies deployed by the factory as soon as a newer strategy is deployed for the same vault. This makes the factory registry an incomplete record of its deployments and may cause integrations relying on `isDeployedStrategy` to reject legitimate strategies.

Impact

Low.

Recommendation

Track every strategy deployed for each vault using OpenZeppelin's `EnumerableSet.AddressSet`:

```
+import {EnumerableSet} from "@openzeppelin/contracts/utils/structs/
EnumerableSet.sol";

 contract StrategyFactory {
+    using EnumerableSet for EnumerableSet.AddressSet;

-    mapping(address => address) public deployments;
+    mapping(address => EnumerableSet.AddressSet) private deployments;
```

```
9       function newStrategy(
10          address _asset,
11          string calldata _name,
12          address _vault,
13          address _router
14      ) external virtual returns (address) {
15          // Strategy deployment and configuration omitted.

17  -       deployments[_vault] = address(_newStrategy);
18  +       deployments[_vault].add(address(_newStrategy));
19          return address(_newStrategy);
20      }

22      function isDeployedStrategy(address _strategy) external view returns (bool)
    {
23          address _vault = IStrategyInterface(_strategy).vault();
24  -       return deployments[_vault] == _strategy;
25  +       return deployments[_vault].contains(_strategy);
26      }
27  }
```

Consider also exposing getter methods for the number of strategies and the strategy at a given index so that integrations can enumerate all deployments associated with a vault.

Developer Response

Agreed. Fixed in PR#6.

2.8 Gas Savings Findings

2.8.1 Factory `newStrategy` methods perform redundant external calls

Technical Details

After deploying a new strategy, both `StrategyFactory.newStrategy` and `CurvanceOptimizerFactory.newStrategy` execute four sequential external calls to configure the deployed contract's privileged roles:

```
strategy.setPerformanceFeeRecipient(performanceFeeRecipient);
strategy.setKeeper(keeper);
strategy.setPendingManagement(management);
strategy.setEmergencyAdmin(emergencyAdmin);
```

Each call introduces an additional external call, calldata handling, fallback dispatch, and delegatecall overhead. Since these values are always configured together during deployment, performing four separate calls unnecessarily increases the gas cost of every strategy and optimizer deployment.

Impact

Gas optimization.

Recommendation

Implement a single management-restricted method on each strategy implementation that configures the performance fee recipient, keeper, pending management, and emergency administrator while preserving the existing validation and event emission behavior. Both factories can then initialize all four roles through one external call:

```
-         strategy.setPerformanceFeeRecipient(performanceFeeRecipient);
-         strategy.setKeeper(keeper);
-         strategy.setPendingManagement(management);
-         strategy.setEmergencyAdmin(emergencyAdmin);
+         strategy.initializeFactoryAddresses(
+             management,
+             performanceFeeRecipient,
+             keeper,
+             emergencyAdmin
+         );
```

Developer Response

Agreed. Fixed in PR#6.

2.9 Informational Findings

2.9.1 `positionValue` accepts unregistered markets while `marketMaxWithdraw` rejects them

Technical Details

`CurvanceOptimizer` exposes two public per-market read functions that validate their input inconsistently:

```
function positionValue(address market) public view returns (uint256) {
    ICurvanceVault target = ICurvanceVault(market);
    return target.convertToAssets(target.balanceOf(address(this)));
}

function marketMaxWithdraw(address market) public view returns (uint256) {
    address manager = marketManagers[market];
    require(_registered(market), "unknown market");
    ...
}
```

`positionValue` accepts any address. Passed an arbitrary ERC-4626 vault it returns `convertToAssets(balanceOf(strategy))` for a destination the strategy does not manage — a well-formed number carrying no meaning for the strategy's accounting.

The two disagree about the same address after a market is retired: following `removeMarket`, `marketMaxWithdraw(removed)` reverts with `"unknown market"` while `positionValue(removed)` still returns a value for any shares the strategy holds, including dust.

All six internal call sites pass `markets[i]` from the registered array, so no internal path is affected.

Impact

Informational. Both functions are `view`, no internal caller can pass an unregistered address, and no funds are at risk. The concern is that an integrator may reasonably assume `positionValue` validates its input as its sibling does, and act on a value not tied to a registered destination.

Recommendation

Validate on the public function, keeping an unchecked internal path so the registration `SLOAD` is not paid on every loop iteration — `positionValue` is called up to twice per market inside `_withdrawMarket` alone:

```
-    function positionValue(address market) public view returns (uint256) {
+    function positionValue(address market) public view returns (uint256) {
+        require(_registered(market), "unknown market");
+        return _positionValue(market);
+    }
+
+    function _positionValue(address market) private view returns (uint256) {
         ICurvanceVault target = ICurvanceVault(market);
         return target.convertToAssets(target.balanceOf(address(this)));
     }
```

Internal callers in `_invest`, `_positionAssets`, `_withdrawMarket` and `_tend` then use `_positionValue`.

Developer Response

Agreed. Fixed in PR#6.

2.9.2 `CurvanceOptimizer` does not override `availableDepositLimit`

Technical Details

`CurvanceOptimizer` inherits `BaseHealthCheck.availableDepositLimit`, which returns `type(uint256).max` once a depositor is allowlisted. Deposits are therefore accepted well beyond the sum of configured `allocationTargets`; `_invest` fills each market only up to its target and the remainder stays idle.

Idle assets are still counted by `_harvestAndReport`, so they participate in price-per-share while earning nothing, diluting the yield of existing holders until management raises targets or the excess is withdrawn.

This also differs from the sibling strategy: `Base4626Compounder.availableDepositLimit` caps deposits at `vault.maxDeposit(address(this))`, so `CurvanceLender` cannot take in more than its destination will accept.

Impact

Informational. Deposit access is closed by default and the allowlist is expected to contain only the parent vault, so the condition requires management to over-fund the strategy relative to its own targets.

Recommendation

Override `availableDepositLimit` to return the remaining aggregate capacity, so the limit reflects what the strategy can actually deploy or document if unbounded deposits are intentional.

Developer Response

This is intentional, where the behavior is documented in the README and covered by `test_depositsAboveTargetsRemainIdle`. Deployments are intended to restrict deposit recipients to the parent vault, with management coordinating funding and allocation targets.

2.9.3 `addRewardToken` accepts duplicates, which `removeRewardToken` then fails to remove

Technical Details

`CurvanceLender.addRewardToken` validates only that the token is neither the underlying asset nor the vault:

```
function addRewardToken(address _rewardToken) external onlyManagement {
    require(_rewardToken != address(asset) && _rewardToken != address(vault), "Invalid
reward token");
    rewardTokens.push(_rewardToken);
}
```

Because duplicates are reachable, `removeRewardToken` fails to remove them all. It swap-and-pops without a `break` and without re-checking index `i`, so the element moved into that slot is skipped:

```
function removeRewardToken(address _rewardToken) external onlyManagement {
    for (uint256 i = 0; i < rewardTokens.length; i++) {
        if (rewardTokens[i] == _rewardToken) {
            rewardTokens[i] = rewardTokens[rewardTokens.length - 1];
            rewardTokens.pop();
        }
    }
}
```

Adding `[r1, r2, r1]` and then calling `removeRewardToken(r1)` leaves `rewardTokens` at length 2 with `r1` still at index 0.

Note that `CurvanceOptimizer._addMarket`'s `!_registered(market)` check does not apply here: it is a different function on a different contract, guarding markets rather than reward tokens, and `CurvanceLender` has no markets.

Impact

Informational. Both functions are `onlyManagement`, and a stale entry causes one additional `_swapFrom` on a zero balance, which is a no-op. The concern is that the function silently fails its stated purpose, so management may believe a token has been delisted when it has not.

Recommendation

Block duplicates on `addRewardToken()`.

Developer Response

Valid. Fixed in PR#6.

2.9.4 `CurvanceOptimizer.emergencyWithdraw` can revert when destinations share underlying liquidity

Technical Details

`CurvanceOptimizer._emergencyWithdraw` sizes its unwind from `availableWithdrawLimit`, which sums `marketMaxWithdraw` over every registered market:

```
uint256 liquid = availableWithdrawLimit(address(this)) - asset.balanceOf(address(this));
_withdraw(Math.min(amount, liquid));
```

`_withdraw` then requires the full amount to be freed, reverting with `"insufficient liquidity"` otherwise.

When a direct market and a `LendingOptimizer` that allocates into that same market are both registered, the two quotes draw on the same underlying cash and the sum double-counts it. The README notes this for `maxWithdraw`, but the same quote also drives the emergency path — so `emergencyWithdraw(type(uint256).max)`, the natural way to request a full unwind, reverts. The recorded AUSD deployment is this configuration: the cAUSD market alongside the hyAUSD `LendingOptimizer`, which itself allocates into cAUSD.

Impact

Informational. Only a `type(uint256).max` request reverts. `_withdraw` re-reads `marketMaxWithdraw` per market inside the loop, so any amount at or below true joint liquidity succeeds and partial exits can be repeated, and RELEASE.md already directs operators to simulate and pass an explicit amount.

The residual cost is that the contract cannot report a workable amount — `availableWithdrawLimit` is itself the overstated quote — so it must be derived from off-chain Curvance state or by trial during incident response.

Recommendation

Document this case in the function natspec.

Developer Response

Valid. Fixed in PR#6

2.9.5 `removeMarket` can be blocked by donating dust shares to the optimizer

Technical Details

`CurvanceOptimizer.removeMarket` requires the strategy's raw share balance in a destination to be exactly zero:

```
require(ICurvanceVault(market).balanceOf(address(this)) == 0, "position remains");
```

Destination shares are ordinary ERC-20 tokens, so anyone can transfer them to the optimizer without calling any of its functions. A donated share worth zero underlying assets cannot then be burned: `_tend` exits only positions whose `positionValue` exceeds the allocation target, `_withdraw` skips any market whose `marketMaxWithdraw` is zero, and no management function redeems raw shares independently of their asset value. Because removal is a public transaction, an observer can also front-run each attempt with another dust transfer.

Precondition. A share is worth zero assets only while the destination's exchange rate is below 1:1. `BaseCToken` and `LendingOptimizer` both disable virtual shares and use a zero decimals offset, and `_initialConvertToShares` mints 1:1, so `convertToAssets(1)` reduces to `floor(totalAssets / totalSupply)` — zero only when `totalAssets < totalSupply`, meaning the destination has been impaired. At or above 1:1 a donated share is worth at least one asset unit, `tend()` clears it as surplus over a zero target, and removal succeeds.

Impact

Informational. Requires an already-impaired destination, and the condition resolves on its own once accrual restores the exchange rate.

While it holds, management cannot retire the affected market: it remains part of `_accrue`, `_positionAssets`, withdrawal-limit computation and reporting, and at the 16-market cap a blocked removal also prevents registering a replacement. No user funds can be stolen or lost.

Recommendation

Test value rather than raw share count:

```
-        require(ICurvanceVault(market).balanceOf(address(this)) == 0, "
position remains");
+        uint256 shares = ICurvanceVault(market).balanceOf(address(this));
+        require(ICurvanceVault(market).previewRedeem(shares) == 0, "position
remains");
```

The stated invariant — that removal must not hide remaining value — is preserved, since a position worth zero assets hides nothing. This still leaves a front-run window for donations that do carry value; closing that would require an atomic exit-and-remove, which is likely disproportionate given the precondition.

Developer Response

Agreed. Fixed in PR#6.

2.9.6 Missing event emission in factory `setAddresses` methods

Technical Details

Both `StrategyFactory.setAddresses` and `CurvanceOptimizerFactory.setAddresses` allow the current management address to update the management, performance fee recipient, and keeper addresses assigned to future deployments.

However, neither function emits an event when these privileged configuration values are modified. Consequently, off-chain monitoring systems cannot reliably track these changes without inspecting contract storage or transaction calldata.

Impact

Informational.

Recommendation

Define an `AddressesUpdated` event in both factories and emit it within each external `setAddresses` method:

```
+event AddressesUpdated(
+    address indexed management,
+    address indexed performanceFeeRecipient,
+    address indexed keeper
+);
```

```
 function setAddresses(address _management, address _performanceFeeRecipient,
address _keeper) external {
     require(msg.sender == management, "!management");
     management = _management;
     performanceFeeRecipient = _performanceFeeRecipient;
     keeper = _keeper;
+    emit AddressesUpdated(_management, _performanceFeeRecipient, _keeper);
 }
```

For `CurvanceOptimizerFactory`, which delegates the storage updates to `_setAddresses`, emit the event only from the external method so construction does not produce a misleading configuration-update event:

```
 function setAddresses(address _management, address _performanceFeeRecipient,
address _keeper) external {
     require(msg.sender == management, "!management");
     _setAddresses(_management, _performanceFeeRecipient, _keeper);
+    emit AddressesUpdated(_management, _performanceFeeRecipient, _keeper);
 }
```

Developer Response

Agreed. Fix in PR#6

yAudit

Curvance Lender Strategy

Completed 2026-09-11