certora

Security Assessment & Formal Verification Report

Tokemak

January 2024

Prepared for
Tokemak

certora TOKEMAK

Table of content

Project Summary

Project Scope

Repo Name	Repository	Commits	Compiler version	Platform
Tokemak-v2-core	https://github.com/Certora/Tokemak-v2-core (private)	d30d6e6	Solidity 0.8.17	EVM

Project Overview

This document describes the specification and verification of the **Tokemak v2 LMPStrategy contract** using the Certora Prover and manual code review findings. The work was undertaken from **2 January 2023** to **5 February 2024**.

The following contract list is included in our scope:

`src/strategy/LMPStrategy.sol`

However, as that contract does not stand on its own and is only a part of a larger system, it was necessary to also partially look at some other contracts (namely `LMPVault.sol` and `LMPDebt.sol`). Any issues found on these contracts are also mentioned in this report.

The Certora Prover demonstrated the implementation of the Solidity contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts. During the verification process and the manual audit, the Certora Prover discovered bugs in the Solidity contracts code, as listed below.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Acknowledged	Code Fixed
Critical	1	1	1
High	0	0	0
Medium	0	0	0
Low	3	3	2
Informational	7	7	4
Total	12	12	7

Detailed Findings

Critical Severity Concerns

C-1. Moving Idle funds to a Destination Vault always reverts

Impact: High
Probability: High
Description:
When the `verifyRebalance()` function is being called with the `params.destinationOut` parameter set to `address(lmpVault)` (which is the case whenever funds are being transferred out from the "Idle" state), the function reverts. This is because:

1. the `getRebalanceValueStats()` function, which would immediately be called, would try to call the `getValidatedSpotPrice()` function (LMPStrategy.sol#L375) using the `IDestinationVault` interface, and as that function is not implemented in the LMPVault.sol contract, the transaction would revert.

2. the `verifyLSTPriceGap()` function, which would immediately be called, would try to call the `underlyingTokens()` (LMPStrategy.sol#L412) and `getPool()` (LMPStrategy.sol#L414) functions using the `IDestinationVault` interface, and as those functions are not implemented in the LMPVault.sol contract, the transaction would revert.

Tokemak's response: There is a check currently being performed for the "in" destination that we'll apply to the "out" as well. We should not be doing the price check in this scenario.

Low Severity Concerns

L-1 Loss of precision

Impact: Low
Probability: High
Description:
The variable `swapOffsetPeriod` represents time in days. As any non-whole number cannot be represented, the expression `swapOffsetPeriod = swapOffsetPeriod / 2` (LMPStrategy.sol#L287) would result in a loss of precision of half a day for any odd number of days originally represented in the `swapOffsetPeriod` variable.

Tokemak's response: Truncating is the right (conservative) choice but can be made more explicit by casting.

L-2. Incorrect evaluation of an edge case

Impact: Medium
Probability: Low
Description:
The `verifyCleanUpOperation()` function should evaluate whether the position is a "dust position", i.e. less than 2% of total assets are not in idle (LMPStrategy.sol#L499). In the edge case in which `lmpVault.totalAssets()` is very small, the function might not work as intended. For example, if `lmpVault.totalAssets()` is less than 50, the condition can never be satisfied even if `currentDebt` is 0. It might be better to use the `<=` operator instead for such use case.

Tokemak's response: The case where totalAssets are < 50 ETH where RHS of the condition evaluates to 0, we will handle separately. Going to add an additional check for this condition.

```
Unset
// If the current position is < 2% of total assets, trim to idle is allowed
       if ((currentDebt < lmpVault.totalAssets() / 50) || (currentDebt < 10)) {
           return true;
       }
```

L-3. Division by zero not prevented

Impact: Low
Probability: Low
Description:
The divisions below take an input parameter that does not have any zero-value checks, which may lead to the functions reverting when zero is passed.

```
Unset
# File: src/strategy/LMPStrategy.sol
LMPStrategy.sol:392: uint256 slippage = swapCost * 1e18 / outEthValue;
LMPStrategy.sol:420: if (((priceSafe * 1.0e18 / priceSpot - 1.0e18) * 10_000)
/ 1.0e18 > tolerance) {
LMPStrategy.sol:436: if (((priceSpot * 1.0e18 / priceSafe - 1.0e18) * 10_000)
/ 1.0e18 > tolerance) {
LMPStrategy.sol:496: uint256 currentDebt = destInfo.currentDebt *
currentShares / destInfo.ownedShares;
LMPStrategy.sol:523: uint256 currentDebt = destInfo.currentDebt *
currentShares / destInfo.ownedShares;
LMPStrategy.sol:538: return destValueAfterRebalance * 1e18 /
lmpAssetsAfterRebalance >= trimAmount;
LMPStrategy.sol:929: (uint40(block.timestamp) - lastRebalanceTimestamp) / 1
days / uint40(swapCostOffsetRelaxThresholdInDays);
```

Tokemak's response: We will add div by 0 checks for some of these.

Informational Concerns

I-1. Possibly unreachable code

In order for the if statement (LMPStrategy.sol#L633) to be satisfied, the `getDestinationSummaryStats()` function must be called with the `destAddress` parameter set to `address(lmpVault)`. The function could be called either by the `getRebalanceInSummaryStats()` function, which is an internal function, or by the `getRebalanceOutSummaryStats()` function, which is an external function.

In the case of `getRebalanceInSummaryStats()`, in order for the if statement to be satisfied it must be called with `rebalanceParams.destinationIn` set to `address(lmpVault)`. The function could only be called by the `verifyRebalance()` function, which specifically checks whether `params.destinationIn == address(lmpVault)` (line 261) and returns without calling `getRebalanceInSummaryStats()` if that condition holds.

In the case of `getRebalanceOutSummaryStats()`, if the function is called by the `LMPVault` contract and the if statement is true, then the return value `outSummary` would be passed to the `flashRebalance()` function in the `LMPDebt` contract, which would pass it to the `verifyRebalance()` function in the strategy contract, which (if it is the `LMPStrategy` contract) does not use any of the values set to `outSummary` if the original if statement is true, but does use the `outSummary.compositeReturn` variable, which would be 0 as it was never set.

Tokemak's response: The compositeReturn always being 0 is intentional in the scenario described. The flow here is when we are taking idle assets and deploying them out to the market. The idle asset for this vault is WETH which has no return on its own, so always 0.

I-2. TODO Left in the code

TODO comments should not be in the final version. Unfinished tasks should be completed and then deleted from the comments.

Tokemak's response: Acknowledged and those tasks have been completed.

I-3. decimals() is not a part of the ERC-20 standard

The `decimals()` function is not a part of the ERC-20 standard, and was added later as an optional extension. As such, some valid ERC20 tokens do not support this interface, so it is unsafe to blindly cast all tokens to this interface, and then call this function.

```
Unset
# File: src/strategy/LMPStrategy.sol
LMPStrategy.sol:368: uint8 tokenOutDecimals =
IERC20Metadata(params.tokenOut).decimals();
LMPStrategy.sol:369: uint8 tokenInDecimals =
IERC20Metadata(params.tokenIn).decimals();
LMPStrategy.sol:744: uint256 rewardDivisor = 10 **
IERC20Metadata(rewardToken).decimals();
LMPStrategy.sol:779: uint256 lpTokenDivisor = 10 **
IDestinationVault(destAddress).decimals();
```

Tokemak's response: Acknowledged.

I-4. Event is never emitted

The following are defined but never emitted.

```
Unset
# File: src/strategy/LMPStrategy.sol
LMPStrategy.sol:153: event RebalanceComplete();
```

Tokemak's response: Acknowledged. This has been cleaned up.

I-5. public functions not called by the contract should be declared external instead

```
Unset
# File: src/strategy/LMPStrategy.sol
LMPStrategy.sol:251: function verifyRebalance()
```

Tokemak's response: Acknowledged. This will be addressed.

I-6. Signature use at deadlines should be allowed

According to EIP-2612, signatures used on exactly the deadline timestamp are supposed to be allowed. While the signature may or may not be used for the exact EIP-2612 use case (transfer approvals), for consistency's sake, all deadlines should follow this semantic. If the timestamp is an expiration rather than a deadline, consider whether it makes more sense to include the expiration timestamp as a valid timestamp, as is done for deadlines.

```
Unset
# File: src/strategy/LMPStrategy.sol
LMPStrategy.sol:754: if (periodFinish > block.timestamp) {
LMPStrategy.sol:765: periodFinish >= block.timestamp + 7 days
LMPStrategy.sol:766: || (hasCredits && periodFinish > block.timestamp + 3 days)
```

Tokemak's response: Acknowledged. This will be fixed in the following release.

I-7. Solidity version 0.8.20 may not work on other chains due to PUSH0

The compiler for Solidity 0.8.20 switches the default target EVM version to Shanghai, which includes the new `PUSH0` op code. This op code may not yet be implemented on all L2s, so deployment on these chains will fail. To work around this issue, use an earlier EVM version. While the project itself may or may not compile with 0.8.20, other projects with which it integrates, or which extend this project may, and those projects will have problems deploying these contracts/libraries.

Tokemak's response: Acknowledged. We are on 0.8.17 in the current repo.

Gas Optimizations Recommendations

G-1. Inefficient calculation

The `verifyRebalanceToIdle()` (LMPStrategy.sol#L445) function sets the `maxSlippage` variable to be the maximal value out of several possible values based on certain conditions. In each scenario, an if statement checks whether a condition holds and whether the corresponding value is greater than the value that was calculated so far, and only if both of these hold then the current value is being updated. A more efficient way to achieve the same thing might be to switch the order of the two conditions in each of the if statements. As the check of whether the new "candidate value" is greater than the current value of `maxSlippage` is cheaper (and also possibly more likely to be false), the lazy evaluation of the expression inside the if statement might result in the saving of some units of gas, as the other condition, which is a more gas-consuming function call, might not be needed to be evaluated.

Tokemak's response: Acknowledged. This will be fixed.

G-2. Redundant multiplication

The expression `priceSafe * 1.0e18 / priceSpot - 1.0e18) * 10_000) / 1.0e18` (LMPStrategy.sol#L420) could be rewritten using fewer multiplications, as the original expression multiplies and divides all elements by `10**18`.
The expression `priceSafe * 10_000 / priceSpot - 10_000` should be equivalent for all values which do not cause the original expression to overflow.

Tokemak's response: Acknowledged.

G-3. Unchecking arithmetics operations that can't underflow/overflow

Solidity version 0.8+ comes with implicit overflow and underflow checks on unsigned integers. When an overflow or an underflow isn't possible (as an example, when a comparison is made before the arithmetic operation), some gas can be saved by using an unchecked block: https://docs.soliditylang.org/en/v0.8.10/control-structures.html#checked-or-unchecked-arithmetic

Saves 25 gas per instance

Affected code:

```
Unset
# File: src/strategy/LMPStrategy.sol

LMPStrategy.sol:946: newSwapCostOffset = currentSwapOffset -
swapCostOffsetTightenStepInDays;

LMPStrategy.sol:963: return uint40(block.timestamp) - lastPausedTimestamp <=
pauseRebalanceInSeconds;

LMPStrategy.sol:998: return self - other;

LMPStrategy.sol:1003: return self - other;
```

Tokemak's response: Acknowledged.

G-4. Use calldata instead of memory for function arguments that do not get mutated

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. Each iteration of this for-loop costs at least 60 gas (i.e. `60 * <mem_array>.length`). Using `calldata` directly bypasses this loop.

If the array is passed to an `internal` function which passes the array to another internal function where the array is modified and therefore `memory` is used in the `external` call, it's still more gas-efficient to use `calldata` when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one.

Saves 60 gas per instance.

Affected code:

```
Unset
# File: src/strategy/LMPStrategy.sol

LMPStrategy.sol:252: IStrategy.RebalanceParams memory params,

LMPStrategy.sol:253: IStrategy.SummaryStats memory outSummary

LMPStrategy.sol:593: function
getRebalanceOutSummaryStats(IStrategy.RebalanceParams memory
rebalanceParams)

LMPStrategy.sol:871: function
rebalanceSuccessfullyExecuted(IStrategy.RebalanceParams memory params)
external onlyLMPVault {
```

Tokemak's response: Acknowledged.

G-5. Use shift right/left instead of division/multiplication if possible

While the DIV / MUL opcode uses 5 gas, the SHR / SHL opcode only uses 3 gas. Furthermore, beware that Solidity's division operation also includes a division-by-0 prevention which is bypassed using shifting. Eventually, overflow checks are never performed for shift operations as they are done for arithmetic operations. Instead, the result is always truncated, so the calculation can be unchecked in Solidity version 0.8+

- Use `>> 1` instead of `/ 2`
- Use `>> 2` instead of `/ 4`
- Use `<< 3` instead of `* 8`
- ...
- Use `>> 5` instead of `/ 2^5 == / 32`
- Use `<< 6` instead of `* 2^6 == * 64`

Saves around 2 gas + 20 for unchecked per instance.

Affected code:

```
Unset
# File: src/strategy/LMPStrategy.sol

LMPStrategy.sol:287: swapOffsetPeriod = swapOffsetPeriod / 2; // TODO: this should be configurable

LMPStrategy.sol:837: scalingFactor -= scalingFactor * timeSinceDiscountSec / halfLifeSec / 2;
```

Tokemak's response: Acknowledged. This will be fixed.

G-6. Increments/decrements can be unchecked in for-loops

In Solidity 0.8+, there's a default overflow check on unsigned integers. It's possible to uncheck this in for-loops and save some gas at each iteration, but at the cost of some code readability, as this uncheck cannot be made inline: [additional information].

```
Unset
- for (uint256 i; i < numIterations; i++) {
+ for (uint256 i; i < numIterations;) {
  // ...
+   unchecked { ++i; }
}
```

These save around 25 gas saved per instance. The same can be applied with decrements (which should use `break` when `i == 0`). The risk of overflow is non-existent for `uint256`.

Affected code:

```
Unset
# File: src/strategy/LMPStrategy.sol
LMPStrategy.sol:415: for (uint256 i = 0; i < numLsts; ++i) {
LMPStrategy.sol:431: for (uint256 i = 0; i < numLsts; ++i) {
LMPStrategy.sol:553: for (uint256 i = 0; i < numLsts; ++i) {
LMPStrategy.sol:582: for (uint256 i = 0; i < len; ++i) {
LMPStrategy.sol:655: for (uint256 i = 0; i < numLstStats; ++i) {
LMPStrategy.sol:733: for (uint256 i = 0; i < numRewards; ++i) {
LMPStrategy.sol:808: for (uint256 i = 0; i < numLsts; ++i) {
LMPStrategy.sol:828: for (uint256 j = 1; j < len; ++j) {
```

Tokemak's response: Acknowledged.

Formal Verification

Assumptions and Simplifications Made During Verification

General Assumptions

A. Any loop was unrolled to two iterations at most.

Functions summarization

When a call to functions `getPriceInEth(address)`, `getBptIndex()` and `getSpotPrice(address, address, address)` occurred, instead of using a contract implementation, we used a ghost mapping to retrieve an arbitrary return value that does not change during a block, but can vary from block to block.

Missing implementation

Functions `addToWithdrawalQueueHead(address)` and `addToWithdrawalQueueTail(address)` have no implementation and always revert which blocks some code paths. It was agreed with Tokemak team that these functions remain unimplemented but revert command is removed to allow exploration of all system flows.

Formal Verification Properties

Notations

✅ Indicates the rule is formally verified.
❌ Indicates the rule is violated.

Since the protocol consists of different contracts, we will present the relative properties for each of the main contracts in separate sections.

The following files were formally verified, and the properties are listed below per library/contract:

A. LMPStrategy.sol

LMPStrategy.sol

Assumptions

- We verified the contract functions against an arbitrary storage state.

Properties

1. ✅ Offset period should be between `swapCostOffsetMaxInDays` and `swapCostOffsetMinInDays`. (*offsetIsInBetween*)
2. ✅ If rebalance is successful and strategy is paused, `verifyRebalance()` must execute `verifyRebalanceToIdle()`. (*pausedStrategyVerifiesIdle*)
3. ✅ If rebalance is successful and `verifyRebalanceToIdle()` was not executed, strategy is not paused. (*successfulNotPaused*)
4. ✅ `violationTrackingState.len` cannot exceed 10. (*noVioLenThanTen*)
5. ✅ `violationTrackingState.violationCount` can be increased only by 1 at a time. (*cantJumpTwoViolationsAtOnce*)
6. ✅ `navTrackingState.lastFinalizedTimestamp` cannot be in the future. (*navTimestampNotInFuture*)
7. ✅ `navTrackingState.lastFinalizedTimestamp` can't be decreased (*navTimestampCantBeDecreased*)
8. ✅ `navTrackingState.currentIndex` can't exceed 90 (*currentIndexInBetween*)
9. ✅ `navTrackingState.len` can be increased only by 1 at a time (*cantJumpTwoNavTracksAtOnce*)
10. ✅ `navTrackingState.len` can only be between 0 and 91 (*currentLenInBetween*)
11. ✅ `lastRebalanceTimestamp` can't be in the future (*lastRebalanceTimestampNotInFuture*)
12. ✅ `lastRebalanceTimestamp` can't be decreased (*lastRebalanceTimestampCantBeDecreased*)
13. ❌If rebalance is successful:
    a. ✅ `destinationIn` isn't equal `destinationOut`
    b. ✅ if `destinationIn == LMPVault` then `tokenIn == baseAsset`
    c. ❌ if `destinationOut == LMPVault` then `tokenOut == baseAsset`
    d. ✅ if `destinationIn != LMPVault` then `tokenIn == destinationIn.underlying`
    e. ❌ if `destinationOut != LMPVault` then `tokenOut == destinationOut.underlying`

    (*rebalanceSuccess*)

    f. ✅ `destinationIn/destinationOut` should be registered, queued for removal or be a `LMPVault` (*inOutRegRemovalOrVault*)

c and e can't be fully verified because of the issue C-1.

14. ❌ There exists a scenario when if `destinationOut == LMPVault` then `verifyRebalance()` doesn't revert
(*checkRevertCase*)

    The property is violated and proves unreachability of the case when `destinationOut == LMPVault` described in issue C-1.

15. ✅ Slippage of a successful rebalance is always <= `maxNormalOperationSlippage`

    However, a corollary of the above property fails due to the rounding down that happens in the code while calculating the slippage.

```
Unset
ethValueOut <= 10^18 * ethValueIn /(10^18 - maxNormalOperationSlippage)
```

    A modified version of the property was verified.

```
Unset
assert ethValueOut <= 10^18 * ethValueIn /(10^18 - (allowedSlippage+1))
```

    (*slippageLeMaxSlippage*)

16. ✅ if X of the last 10 rebalances were violations, offset should decrease
(*tighteningOffset*)
    a. Assuming that `_swapCostOffsetPeriod` is greater or equal to `swapCostOffsetMinInDays()`
    b. Values range for some immutable state variables was constrained according to the list provided by Tokemak team. Particularly:
        i. 7 <= `swapCostOffsetInitInDays` <= 90;
        ii. 1 <= `swapCostOffsetTightenThresholdInViolations` <= 10;
        iii. 1 <= `swapCostOffsetTightenStepInDays` <= 7;
        iv. 14 <= `swapCostOffsetRelaxThresholdInDays` <= 90;
        v. 1 <= `swapCostOffsetRelaxStepInDays` <= 7;
        vi. 7 <= `swapCostOffsetMaxInDays` <= 90;
        vii. 7 <= `swapCostOffsetMinInDays` <= 90;
        viii. 7 <= `_swapCostOffsetPeriod` <= 90;
        ix. `swapCostOffsetMaxInDays` > `swapCostOffsetMinInDays` ;

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.