SMART CONTRACTS REVIEW

zokyo

May 14th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING MAXAPY

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: 0 points

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 1 High issue: 1 resolved = 0 points deducted
- 7 Medium issues: 7 resolved = 0 points deducted
- 5 Low issues: 5 resolved = 0 points deducted
- 6 Informational issues: 6 resolved = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

This document outlines the overall security of the maxAPY smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the maxAPY smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the maxAPY team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the maxAPY repository:
Repo: https://github.com/VerisLabs/SharePriceOracle

Last commit - deb07d86c5047b75bf4ac4492d9ba1ae49326590

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- src/
  - MaxLzEndpoint.sol
  - SharePriceRouter.sol
  - adapters
    - Api3.sol
    - Chainlink.sol
  - interfaces
    - IERC20.sol
    - IERC20Metadata.sol
    - IERC4626.sol
    - ILayerZeroEndpointV2.sol
    - ILayerZeroReceiver.sol
    - IMessageLibManager.sol
    - IMessagingChannel.sol
    - IMessagingComposer.sol
    - IMessagingContext.sol
    - ISharePriceRouter.sol
    - api3
      - IProxy.sol
    - chainlink
      - IChainlink.sol
  - libs
    - Bytes32Helper.sol
    - MsgCodec.sol
    - base
      - BaseOracleAdapter.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of maxAPY smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contract by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

SharePriceOracle is a decentralized oracle system for sharing ERC4626 vault share prices across different L2 networks using LayerZero. It is a multi-adapter oracle system that supports multiple price feeds and fallback mechanisms.
SharePriceRouter contract is the main contract that manages multiple oracle adapters and provides unified price conversion. MaxLzEndpoint contract handles cross-chain communication through LayerZero protocol.
Also there are multiple adapters such as Chainlink, Api3, Balancer, UniswapV3 etc. each having specific adapter contract fetching and verifying the assets prices.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Name of company team and the Name of company team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Incorrect getAmountOut(..) use in Aerodrome v1	High	Resolved
2	_lzSend() may revert if msg.value == 0 & address(this).balance > 0	Medium	Resolved
3	Not checking for stale prices	Medium	Resolved
4	Not checking for down L2 sequencer	Medium	Resolved
5	Unnecessary value encoded and incorrect decoding	Medium	Resolved
6	Incorrect timestamp returned for latest share prices	Medium	Resolved
7	Method isSupportedAsset(...) might return wrong answer	Medium	Resolved
8	Missing reentrancy check	Medium	Resolved
9	Revoke ENDPOINT_ROLE from the previous lzEndpoint	Low	Resolved
10	Use abi.encode(...) instead of abi.encodePacked(...) to mitigate ambiguous encoding	Low	Resolved
11	ETH might get stuck in the MaxLzEndpoint contract	Low	Resolved
12	Unsafe casting to uint240	Low	Resolved
13	Balancer staleness block check might get reverted with underflow	Low	Resolved
14	Assign chainId using block.chainId	Informational	Resolved
15	Ambiguous event name	Informational	Resolved
16	Wrong comment	Informational	Resolved
17	Method stringToBytes32 may return garbage data	Informational	Resolved
18	Confusing comment	Informational	Resolved
19	Price manipulation as spot price is read for Aerodrome v1	Informational	Resolved

Incorrect getAmountOut(..) use in Aerodrome v1

In AerodromeV1.sol adapter contract, in the method getPrice(), it calculates the amount of base token out for 1 unit of asset input as follows:

```
        uint256 price = IAerodromeV1Pool(data.pool).getAmountOut(uint256(1
* (10 ** data.quoteTokenDecimals)), asset); //@audit wrong tokenIn
```

Here, it passes the amountIn in term of quoteToken instead of the asset decimals as amountIn and tokenIn should always be for same token.

This will return incorrect output amount in terms of base token. Further this price is multiplied by base token price in USD/ETH to calculate the asset token price in USD/ETH which will be incorrect too.

Recommendation:

Update amountIn to be (1 * (10 ** asset.decimals)) to get amount in base token.

_lzSend() may revert if msg.value == 0 & address(this).balance > 0

In Contract MaxLzEndpoint.sol, the method _handleABAResponse(...) decodes the vault addresses, gets appropriate vault reports, and sends them back to the source chain. The fee required to send the vault report successfully is checked as follows:

```
        if (address(this).balance < fee.nativeFee) revert
InsufficientFunds();
```

This condition will pass if ETH is present in the contract or sent as msg.value. But further in the _lzSend(...) method, fee is again checked as follows:

```
        if (msg.value < _fee.nativeFee) revert InvalidMessageValue();
```

Here, it is needed that the msg.value for this call must be equal to or greater than the fee required even if the required ETH is present in the contract.
This conflict in check may revert the txn which can be avoided.

Recommendation:

Update the logic to mitigate this scenario.

MEDIUM-2 | RESOLVED

Not checking for stale prices

In Library ChainlinkLib, the method getPrice(...) calls chainlink price feed contracts to get the latest price for assets. In case the returned price is staled, using it can result in potential loss of funds for the protocol.
It should be checked here that the updatedAt returned from latestRoundData is compared with the stateness threshold. Stateness threshold should correspond to the heartbeat of the oracle's price feed which can be found for every price feed available in the chainlink docs here. Once heartbeat is ensured, the staleness of the price can be checked as follows:

```
if (updatedAt < block.timestamp - 60 * 60 /* 1 hour */) {  // considering heartbeat value is 1 hour
  revert("stale price feed");
}
```

Recommendation:
Update the logic to check the price staleness by using the heartbeat value provided for each price feed. It is to be noted that not all price feeds will have the same heartbeat value, hence same heartbeat value must not be used for all price feeds for price staleness calculation.

MEDIUM-3 | RESOLVED

Not checking for down L2 sequencer

In Library OraclePriceLib, the method getPrice(...) calls priceFeed.latestRoundData(...) to get the latest price for assets. Since this oracle contract will be used on L2 layer, it is important to check if the sequencer are down or not to avoid stale pricing data that appears as fresh.

The official chainlink documentation for the same is provided here.

Recommendation:
Use sequencer oracle to determine whether the sequencer is offline or not, and don't allow orders to be executed while the sequencer is offline.

Unnecessary value encoded and incorrect decoding

In Contract MsgCodec.sol, method encodeVaultAddresses(...) does the following to encode vault addresses along with other data.

```
        return abi.encode(_msgType, _message,rewardsDelegate
,_extraReturnOptions.length, _extraReturnOptions, extraOptionsLength);
//@audit why last param?
```

Here, extraReturnOptions' length is encoded twice, of which last one is not required while decoding.

Similarly method encodeVaultReports(...) does the following to encode vault reports with other data.

```
        return abi.encode(_msgType, _reports, extraOptionsLength,
_extraReturnOptions, extraOptionsLength); //@audit why last param encoded?
```

Once redundant encoding is fixed, extraOptionsStart will need to be fixed as follows:

extraOptionsStart = HEADER_SIZE + EXTRA_OPTION_SIZE + (message.length * 32);
now encoding will be
< msgType (32), _message offset (32), rewardDelegate (32), optionsLengthThatWeSet (32), options offset (32), _message length (32), All addresses in the array (N * 32), options length (32), options >
so extraOptionsStart should be (32 + 32 + 32 + 32 + 32 + 32 + 32) + (N * 32) = 7 * 32 + N * 32
currently it is 5 * 32 + 3 * 32 + N * 32

similarly for decodeVaultReports
it will be:
< msgType (32), _reports offset (32), extraOptionsLength (32), options offset (32), _reports length (32), All vault reports (N * 32 * 7), options length (32), extraOptionsData >
so extraOptionsStart should be be (32 + 32 + 32 + 32 + 32 + 32) + (N * 32 * 7).
currently it is 5 * 32 + 3 * 32 + (N * 32 * 7)

Recommendation:

Remove the redundant encoded length values and calculate the extraOptionsStart correctly.

Incorrect timestamp returned for latest share prices

In Contract SharePriceRouter.sol, the method getLatestSharePrice(...) returns a timestamp based on assets (srcAsset and dstAsset). If the stored share price is used, the timestamp when the share price was stored might be earlier than the asset's. This can lead to stale share price being used for further calculation on Vault, which will not be revealed on the vault as wrong timestamp would be used to check the staleness.

Recommendation:

Check if the stored share price's timestamp is earlier than the assets' timestamp and return the oldest timestamp.

Method isSupportedAsset(...) might return wrong answer

In Contract SharePriceRouter.sol, the method isSupportedAsset(...) checks if a local asset is supported or not as follows:
LocalAssetConfig memory config = localAssetConfigs[asset][0];
Here, directly accessing index 0 ([0]) might be problematic as admin can assign any priority level (0,1,2,3...) meaning that Priority 0 might never be assigned.

In case, priority for an asset is not 0 but > 0, isSupportedAsset(...) will return false as config.priceFeed for Priority 0 will be address(0). This will lead to adapters getting wrong answer for the query if asset is supported or not and the adapter getPrice() call will revert.

Recommendation:

Either check for all priorities as below or check specifically for the adapter which is calling the method isSupportedAsset(...).

```
function isSupportedAsset(address asset) external view returns (bool) {
    uint8 highestPriority = assetAdapterPriority[asset];
    for (uint8 i = 0; i <= highestPriority; i++) {
        if (localAssetConfigs[asset][i].priceFeed != address(0)) {
            return true;
        }
    }
    return false;
}
```

MEDIUM-7 | RESOLVED

Missing reentrancy check

In Balancer.sol adapter contract, there is no reentrancy check as values are read from external contracts and Balancer contracts have the possibility for read-only reentrancy attack for methods/contracts mentioned here:
https://forum.balancer.fi/t/reentrancy-vulnerability-scope-expanded/4345

Recommendation:
Add a check to ensure the possibility of no read-only reentrancy.

LOW-1 | RESOLVED

Revoke ENDPOINT_ROLE from the previous lzEndpoint

In Contract SharePriceOracle, the method setLzEndpoint is used to set a new lzEndpoint and grants this new address the ENDPOINT_ROLE as well.

Recommendation:
it is advised to revoke the ENDPOINT_ROLE from the previous lzEndpoint.

LOW-2 | RESOLVED

Use abi.encode(...) instead of abi.encodePacked(...) to mitigate ambiguous encoding

In Contract SharePriceOracle, the method getPriceKey calculates the key as follows:

```
return keccak256(abi.encodePacked(_srcChainId, _vault));
```

Here, before hashing, data is encoded using encodePacked which has a known issue for tightly packing its arguments resulting in the possibility of hash collision.

Recommendation:
Use abi.encode(...) to mitigate this issue.

LOW-3 | RESOLVED

ETH might get stuck in the MaxLzEndpoint contract

In Contract MaxLzEndpoint.sol, the method _handleABAResponse(...) sets address(this) as the refund address as follows:

```
        _lzSend(origin.srcEid, returnMessage, options, fee, address(this));
```

In case, there are refunds, which is possible since we send msg.value >= fee, that ETH will get stuck in the contract if not used for any future call to _lzSend(...).

Recommendation:
Add a method for the owner to withdraw any stuck ETH.

LOW-4 | RESOLVED

Unsafe casting to uint240

In Contract Chainlink.sol, the method _parseData casts the price to uint240 as follows:

```
uint256 newPrice = (uint256(price) * WAD) / (10 ** data.decimals);
pData.price = uint240(newPrice);
```

It is advised here to clamp the value first to type(uint240).max if price > type(uint240).max and then cast the value to uint240.

```
uint256 newPrice = (uint256(price) * WAD) / (10 ** data.decimals);

if (newPrice > type(uint240).max) {

    newPrice = type(uint240).max;

}

pData.price = uint240(newPrice);
```

Similarly for Api3.sol, method _parseData(...) does the following:

```
uint256 rawPrice = uint256(price);

        if (rawPrice > type(uint240).max) {
            rawPrice = rawPrice / 1e9; //@audit why not max uint240?
        }

        pData.price = uint240(rawPrice);
```

Here, if rawPrice is > type(uint240).max, we can clamp and cast rawprice as following:

```
uint256 rawPrice = uint256(price);

        if (rawPrice > type(uint240).max) {
            rawPrice = type(uint240).max
        }

        pData.price = uint240(rawPrice);
```

Recommendation:
Check and update the logic as mentioned above.

LOW-5 | RESOLVED

Balancer staleness block check might get reverted with underflow

In Balancer.sol adapter contract, there is a staleness check for the block in _getPriceFromBalancer() method as follows:

```
(,, uint256 lastChangeBlock) =
balancerVault.getPoolTokens(config.poolId);

        // Convert blocks to approximate time (assuming ~12 second blocks)
        uint256 blockTimeDiff = block.number - lastChangeBlock;
```

Although uncommon, if lastChangeBlock > block.number for any reason, there will be an underflow revert.

Recommendation:
Add the following check to ensure there is no revert.

```
if (approxTimeDiff > config.heartbeat) {
    pData.hadError = true;
    return pData;
}
```

INFORMATIONAL-1 | RESOLVED

Assign chainId using block.chainId

In Contract SharePriceOracle, in the constructor, chainId is assigned as follows:

```
        chainId = _chainId;
```

Recommendation:
Update the logic to assing chainId as follows:

```
        chainId = uint64(block.chainid);
```

INFORMATIONAL-2 | RESOLVED

Ambiguous event name

In Contract MaxLzEndpoint.sol, the method sendSharePrices(...) emits an event as follows:

```
        emit VaultAddressesSent(dstEid, vaultAddresses); //@audit event
name sharePriceSent
```

It mentions vaultAddressSent but this method sends share prices for the already provided vault addresses.

Recommendation:
Update the event name to sharePricesSent(...)

INFORMATIONAL-3 | RESOLVED

Wrong comment

In Contract SharePriceOracle, line#58 mentions a mapping from the key to an array of vault reports which is incorrect.

INFORMATIONAL-4 | RESOLVED

Method stringToBytes32 may return garbage data

In lib Bytes32Helper.sol, the method stringToBytes32(...) converts a string to bytes32. In case the input string is less than 32, it will still return a 32-byte answer of which the remaining data apart from “stringData” will be garbage value arbitrary read from memory which could be `0` as well.

Recommendation:
Add a check to see if length is < 32 and pad the answer with `0`s as follows:

```
// Ensure unused bytes are zeroed out
    if (bytesData.length < 32) {
        result = result & ~(bytes32(uint256(2**(8 * (32 - bytesData.length)) - 1)));
    }
```

INFORMATIONAL-5 | RESOLVED

Confusing comment

In Balancer.sol adapter contract, the formula for the spot price of the asset is commented as follows:
// spotPrice = (balanceQuote / weightQuote) / (balanceAsset / weightAsset)
But calculated as follows:
spotPrice = (balanceQuote * assetWeight) / (balanceAsset * quoteWeight)

Although both are the same, it is advised to update it just for clarity.

INFORMATIONAL-6 | RESOLVED

Price manipulation as spot price is read for Aerodrome v1

In AerodromeV1.sol, getAmountOut() is used to calculate the price for an asset which is vulnerable to price manipulation if liquidity is low.

Recommendation:
Use TWAP if possible.

	MaxLzEndpoint.sol SharePriceRouter.sol Api3.sol Chainlink.sol IERC20.sol IERC20Metadata.sol IERC4626.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	ILayerZeroEndpointV2.sol ILayerZeroReceiver.sol IMessageLibManager.sol IMessagingChannel.sol IMessagingComposer.sol IMessagingContext.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	ISharePriceRouter.sol IProxy.sol IChainlink.sol Bytes32Helper.sol MsgCodec.sol BaseOracleAdapter.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the maxAPY team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the maxAPY team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo