SMART CONTRACTS REVIEW

zokyo

March 31th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract 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 issue: 0 points deducted
- 1 High issue: 1 resolved = 0 points deducted
- 1 Medium issue: 1 resolved = 0 points deducted
- 3 Low issues: 1 resolved and 2 acknowledged = -2 points deducted
- 0 Informational issues: 0 points deducted

Thus, 100 - 2 = 98

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/metavault
Last commit -25567034a5c513777d0063508896a0d973607e92

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

- src/
  - MetaVault.sol
  - RewardDistributor.sol
  - common
    - Lib.sol
    - MetaVaultBase.sol
    - ModuleBase.sol
    - MultiFacetProxy.sol
  - crosschain
    - ERC20Receiver.sol
    - Lib.sol
    - SuperPositionsReceiver.sol
    - SuperformGateway
      - Lib.sol
      - SuperformGateway.sol
      - common
        - GatewayBase.sol
        - Lib.sol
      - modules
        - DivestSuperform.sol
        - InvestSuperform.sol
        - Lib.sol
        - LiquidateSuperform.sol
  - helpers
    - AddressBook.sol
    - Lib.sol
  - interfaces

```
    ├── IBaseRouter.sol
    ├── IERC1155A.sol
    ├── IERC20.sol
    ├── IERC20Metadata.sol
    ├── IERC4626.sol
    ├── IHurdleRateOracle.sol
    ├── IMetaVault.sol
    ├── ISharePriceOracle.sol
    ├── ISuperPositions.sol
    ├── ISuperPositionsReceiver.sol
    ├── ISuperformFactory.sol
    ├── ISuperformGateway.sol
    └── Lib.sol
├── lib
    ├── ERC7540.sol
    ├── Lib.sol
    ├── NoDelegateCall.sol
    └── ReentrancyGuard.sol
├── modules
    ├── AssetsManager.sol
    ├── ERC7540Engine
        ├── ERC7540Engine.sol
        ├── ERC7540EngineReader.sol
        ├── ERC7540EngineSignatures.sol
        ├── Lib.sol
        └── common
            ├── ERC7540EngineBase.sol
            └── ERC7540ProcessRedeemBase.sol
    ├── EmergencyAssetsManager.sol
    ├── Lib.sol
    └── MetaVaultReader.sol
└── types
    ├── DistributorTypes.sol
    ├── ERC7540Types.sol
    ├── Lib.sol
    ├── SuperformTypes.sol
    └── VaultTypes.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/s by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

MetaVault is a next-generation yield aggregation protocol that enables efficient cross-chain capital allocation through asynchronous deposits and withdrawals. Built on the ERC7540 standard, it provides a secure and modular architecture for managing assets across multiple blockchain networks.

MetaVault reimagines cross-chain yield aggregation by implementing:

* ERC7540-compliant asynchronous operations
* Modular proxy architecture for upgradeable components
* Sophisticated withdrawal queue management
* Cross-chain bridging and recovery mechanisms
* Multi-layered fee structure with high watermark

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 maxAPY team and the maxAPY 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	Wrong debt values deducted during divestment	High	Resolved
2	Method notifyRefund may not handle Super position properly	Medium	Resolved
3	Vault more than WITHDRAWAL_QUEUE_SIZE can be added	Low	Resolved
4	Divest amount and Superform Id is not checked for `0`	Low	Acknowledged
5	Remove the Cache.dstVaults[chainIndex][len] if shares == `0`	Low	Acknowledged

Wrong debt values deducted during divestment

In Contract AssetManager.sol, the method divestSingleXChainMultiVault(...) loops through all the vaults for the given chain and update the debt value as following:

```
        for (uint256 i = 0; i < req.superformsData.superformIds.length;) {
            uint256 superformId = req.superformsData.superformIds[i];
            VaultData memory vault = vaults[superformId];
            uint256 sharesBalance = _sharesBalance(vault); //@audit
reducing debt by all shareBalance?
            uint256 sharesValue = vault.convertToAssets(sharesBalance,
asset(), true);
            vault.totalDebt = _sub0(vaults[superformId].totalDebt,
sharesValue).toUint128();
            vaults[superformId] = vault;
            unchecked {
                ++i;
            }
        }
```

Here, the shareBalance is calculated as _shareBalance(vault) which is further defined as:

```
function _sharesBalance(VaultData memory data) internal view returns
(uint256 shares) {
        if (data.chainId == THIS_CHAIN_ID) {
            return ERC4626(data.vaultAddress).balanceOf(address(this));
        } else {
            return gateway.balanceOf(address(this), data.superformId);
        }
    }
```

The actual share amount should be calculated as per the assets amount being withdrawn, not the entire share balance.

Similarly for the method divestMultiXChainMultiVault, wrong debt value is deducted from the vault.debt amount.

Same issue exists for the methods emergencyDivestSingleXChainMultiVault and emergencyDivestMultiXChainMultiVault in the EmergencyAssetManager.sol.

Recommendation:
Update the logic to reduce only the divest amount from vault.debt.

MEDIUM-1 | RESOLVED

Method notifyRefund may not handle Super position properly

In Contract DivestSuperform.sol, the method notifyRefund(...) handles the case when divestment fails and super positions are reminted to the ERC20 receiver contract which further call this method to send the minted Super position to the vault. There is this following check:

```
if (value == 0) return;
```

Here, if there is a super position with value == 0, notifyRefund() will not process it and simply return. This will lead to the super position always stuck in the ERC20 receiver contract.

Recommendation:
Remove the check ` if (value == 0) return`

LOW-1 | RESOLVED

Vault more than WITHDRAWAL_QUEUE_SIZE can be added

In Contract MetaVault.sol, the method addVault(...) adds a new vault and also add the vault in the local withdrawal queue and cross-chain withdrawal queue. Since local withdrawal queue adn cross-chain withdrawal queue has a max limit of WITHDRAWAL_QUEUE_SIZE, there maybe be possibility of vault not being added to any queue but added to the rest mappings.

Recommendation:
Consider adding a check to ensure values are properly added to one of the queue.

LOW-2 | ACKNOWLEDGED

Divest amount and Superform Id is not checked for 0

In Contract DivestSuperform.sol, the method divestMultiXChainSingleVault(...) doesn't check for superformId is 0 or not. Also divest amount is also not checked if it;s 0 or not. Similar issues for the method divestMultiXChainMultiVault in the same contract.

LOW-3 | ACKNOWLEDGED

Remove the Cache.dstVaults[chainIndex][len] if shares == 0

In Contract ERC7540EngineBase.sol, the method _exhaustWithdrawalQueue(...) sets the vault for the local chain or cross-chain as following:

```
            cache.dstVaults[chainIndex][len] = vault.superformId;
```

Further it is checked the amount of shares that can be withdrawal from the vault as following:

```
uint256 shares;
            if (cache.amountToWithdraw >= maxWithdraw) {
                uint256 balance = _sharesBalance(vault);
                shares = balance;
            } else {
                shares = vault.convertToShares(withdrawAssets, asset(),
true);
            }

            if (shares == 0) continue;
```

Here, when shares == 0, loop will continue with the next vault available but previous vault was still added to the cache.

Recommendation:

update the logic to remove the set cache.dstVaults[chainIndex][len]

	MetaVault.sol RewardDistributor.so Lib.sol MetaVaultBase.sol ModuleBase.sol MultiFacetProxy.sol ERC20Receiver.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

	SuperPositionsReceiver.sol SuperformGateway.sol GatewayBase.sol DivestSuperform.sol InvestSuperform.sol LiquidateSuperform.sol AddressBook.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

	IBaseRouter.sol IERC1155A.sol IERC20.sol IERC20Metadata.sol IERC4626.sol IHurdleRateOracle.sol IMetaVault.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

	ISharePriceOracle.sol **ISuperPositions.sol** **ISuperPositionsReceiver.sol** **ISuperformFactory.sol** **ISuperformGateway.sol** **ERC7540.sol** **NoDelegateCall.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

	ReentrancyGuard.sol AssetsManager.sol ERC7540Engine.sol ERC7540EngineReader.sol ERC7540EngineSignatures.sol ERC7540EngineBase.sol ERC7540ProcessRedeemBase.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

	EmergencyAssetsManager.sol MetaVaultReader.sol DistributorTypes.sol ERC7540Types.sol SuperformTypes.sol VaultTypes.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