SMART CONTRACTS REVIEW

zokyo

March 10th 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.

HYPOTHETICAL 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:
- 1 Critical issues: 1 resolved = 0 points deducted
- 2 High issues: 2 resolved = 0 points deducted
- 1 Medium issue: 1 resolved = 0 points deducted
- 2 Low issues: 2 resolved = 0 points deducted
- 3 Informational issues: 3 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 was 1 critical issue 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 - 3f991d084c1c47d337873e8097b6a31191626296

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

AUDITING STRATEGY AND TECHNIQUES APPLIED

```
├── helpers
│   ├── AddressBook.sol
│   └── Lib.sol
├── interfaces
│   ├── IBaseRouter.sol
│   ├── IERC1155A.sol
│   ├── IERC20.sol
│   ├── IERC20Metadata.sol
│   ├── IERC4626.sol
│   ├── IHurdleRateOracle.sol
│   ├── IMetaVault.sol
│   ├── ISharePriceOracle.sol
│   ├── ISuperPositions.sol
│   ├── ISuperformFactory.sol
│   ├── ISuperformGateway.sol
│   └── Lib.sol
├── lib
│   ├── ERC7540.sol
│   ├── Lib.sol
│   ├── NoDelegateCall.sol
│   └── ReentrancyGuard.sol
├── modules
│   ├── AssetsManager.sol
│   ├── ERC7540Engine
│   │   ├── ERC7540Engine.sol
│   │   ├── Lib.sol
│   │   └── common
│   │       └── ERC7540EngineBase.sol
│   └── Lib.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 by industry leaders.
03	Thorough manual review of the codebase line by line.

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	Users' funds can get stuck in Vault using requestDeposit	Critical	Resolved
2	RequestedAssets for divestMultiXChainSingleVault and divestMultiXChainMultiVault is set incorrectly	High	Resolved
3	TotalPendingXChainInvest and PendingXChainInvest not set correctly	High	Resolved
4	Same ROLE_ID set for 2 different roles	Medium	Resolved
5	Approval not removed from previous gateway	Low	Resolved
6	RecoveryAddress is not checked for address(0)	Low	Resolved
7	Method divestSingleDirectMultivault need not to be payable	Informational	Resolved
8	CEI pattern not followed	Informational	Resolved
9	Wrong comment/Natspec	Informational	Resolved

CRITICAL-1 | RESOLVED

Users' funds can get stuck in Vault using requestDeposit

In Contract MetaVault.sol, method requestDeposit calls super.requestDeposit(...) as follows:

```
        requestId = super.requestDeposit(assets, controller, owner);
```

Further, super.requestDeposit(...) does the transfer assets from the owner as following:

```
        asset().safeTransferFrom(owner, address(this), assets);
```

Here anyone can call requestDeposit(...) method passing any user address as owner who has approved assets to the vault. This will transfer funds from user to the vault which will be added for controller which user has not approved and user wont be able to recover these funds as well.

Recommendation:
Update the logic to transfer funds from msg.sender.

HIGH-1 | RESOLVED

RequestedAssets for divestMultiXChainSingleVault and divestMultiXChainMultiVault is set incorrectly

In Contract DivestSuperform.sol, method divestMultiXChainSingleVault and method divestMultiXChainMultiVault set requestedAssets as follows:

```
iata.requestedAssets = totalAmount;
```

But totalAmount is `0` at this moment and set later on. This will set requestedAssets as `0` for both these methods. RequestedAssets value is further used to settle cross-chain divests as follows:

```
        vault.settleXChainDivest(requestedAssets);
```

Which add the requestedAssets value to the totalIdle assets. This will eventually set the totalIdle asset wrong leading to unexpected results.

Recommendation:
Set the requestedAssets correctly in both these methods.

TotalPendingXChainInvest and PendingXChainInvest not set correctly

In Contract InvestSuperform.sol, the method investSingleXChainSingleVault sets the totalPendingXChainInvests as following:

```
totalpendingXChainInvests = amount;
```

It should be incrementing the totalPendingXChainInvests with amount instead it sets totalPendingXChainInvests as amount which is incorrect.

In the same contract, the method investSingleXChainMultiVault, investMultiXChainSingleVault and investMultiXChainMultiVault sets the pendingXChainInvests as following:

```
pendingXChainInvests[superformId] = amount;
```

It should be incrementing the pendingXChainInvests with amount instead it sets pendingXChainInvests as amount which is incorrect.

Recommendation:
Update the logic to add the amount correctly. It would be safer to add than reinitialising with the `amount` value directly.

MEDIUM-1 | RESOLVED

Same ROLE_ID set for 2 different roles

In Contract GatewayBase.sol, there are 2 roles set with the same role id as follows:

```
/// @notice Role identifier for admin privileges
    uint256 public constant ADMIN_ROLE = _ROLE_0; //@audit same role id?
    /// @notice Role identifier for relayer
    uint256 public constant RELAYER_ROLE = _ROLE_0; //@audit same role id?
```

Here one of the role is Admin which has admin privileges and can be maliciously used if assigfned to an unwanted EOA.

Recommendation:

Set different role ids for different roles.

LOW-1 | RESOLVED

Approval not removed from previous gateway

In Contract MetaVault.sol, method setGateway allows Admin to set a new gateway and approve it for asset and superpositions. It doesn't remove the same approval from previous gateway.

Recommendation:

It is advised to remove approvals from previous gateway.

LOW-2 | RESOLVED

RecoveryAddress is not checked for address(0)

In Contract InvestSuperform.sol, the method investSingleXChainSingleVault sets the superpositions receiver address as follows:

```
req.superformData.receiverAddressSP = recoveryAddress;
```

Here, recoveryAddress could be address(0) which will set the receiverAddressSP as address(0) leading to loss of Superposition tokens and loss of user funds.

Recommendation:
Check before assigning if recoveryAddress is address(0) or not.

INFORMATIONAL-1 | RESOLVED

Method divestSingleDirectMultivault need not to be payable

In Contract AssetManager.sol, method divestSingleDirectMultivault is set as payable but it doesn't uses msg.value as there are no gateway transactions.

Recommendation:
Payable can be removed.

CEI pattern not followed

In Contract InvestSuperform.sol, the method investSingleXChainSingleVault does the following:

```
superformRouter.singleXChainSingleVaultDeposit{ value: msg.value }(req);
        uint256 oldPendingAmount = pendingXChainInvests[superformId];
        pendingXChainInvests[superformId] += amount;
        uint256 oldTotalPending = totalpendingXChainInvests;
        totalpendingXChainInvests += amount;
```

Here, external contract is called and then state is updated. Similarly in method investMultiXChainSingleVault, the router contract is called followed by state update which is generally not recommended to mitigate reentracy issues.

Recommendation:
It is advised to update the state first followed by external contract calls.

Wrong comment/Natspec

In Contract InvestSuperform.sol, the method investSingleXChainSingleVault has a comment on line#53 which incorrectly mentions that only owner can call this method.

In Contract InvestSuperform.sol, the method investMultiXChainMultiVault has the following set in the loop:

```
                req.superformsData[i].receiverAddressSP = recoveryAddress;
//@audit can be outside the loop
```

This can be set outside the inside loop as it is set only once for a superformData.

In Contract AssetManager.sol, the method divestSingleDirectSingleVault check if the vault is listed or not as follows:

```
        if (!isVaultListed(vaultAddress)) revert VaultNotListed(); //@audit
check above
```

It should be checked before calculating the the shareBalance and shareValue.

In Contract MetaVault.sol, the method deposit(...) has a wrong natspec comment on line#188 as it mentions shares instead of assets. The same issue on line#196.

In Contract ERC7540Engine, there is an incomplete comment on line#344. Line#423 has a wrong comment as it mentions if it's multivault instead of single vault. Line#650 is missing natspec comment for the last parameter.

	VerisLabs
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