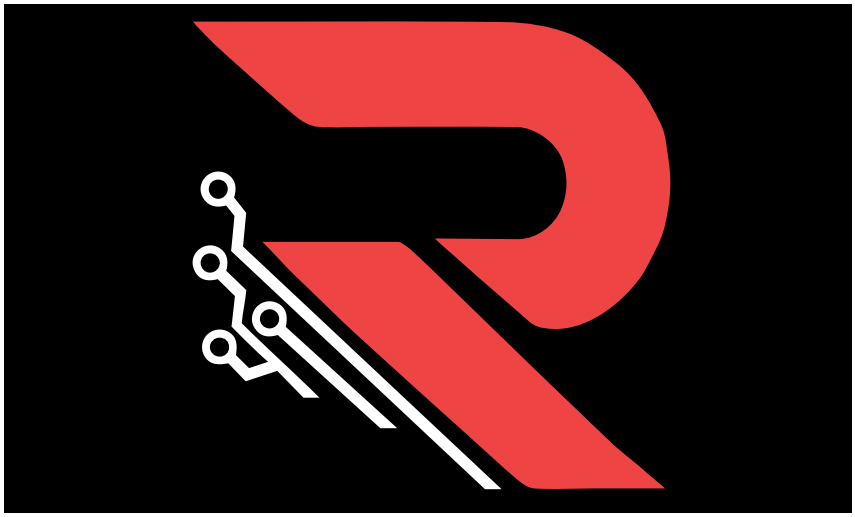
MaxAPY Security Review



Version 2.0

15.02.2025

Conducted by:

MaslarovK, Lead Security Researcher

radev-eth, Lead Security Researcher

Table of Contents

1 About MaslarovK

MaslarovK a Security Reseacher and Co-Founder of Rezolv Solutions.

2 About radev.eth

radev_eth a Security Reseacher and Co-Founder of Rezolv Solutions.

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

4 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	MaxAPY
Repository	https://github.com/VerisLabs/metavault/tree/development
Commit hash	1257f512568eaf96ae14419fedbf2dcdcb767334
Resolution	N/A
Documentation	N/A
Methods	Manual review

Scope

src/MetaVault.sol
src/Modules/ERC7540Engine.sol
src/Modules/AssetsManager.sol
src/CrossChain/SuperPositionsReceiver.sol
src/CrossChain/SuperformGateway.sol
src/CrossChain/DivestSuperform.sol
src/CrossChain/InvestSuperform.sol
src/CrossChain/LiquidateSuperform.sol

Issues Found

Critical risk	2
High risk	1
Medium risk	3
Low risk	4
Informational	3

6 Findings

6.1 Critical risk

6.1.1 In the `MetaVault.sol::chargeGlobalFees()`, the totalIdle is being wrongly increased

Severity: *Critical risk*

Description: In the `chargeGlobalFees`, if the `totalFees > 0`, the `_totalIdle` is being increased using the `_afterDeposit` function, but this must not happen because there is no real funds transfer.

This will mess the accounting and can lead to DoS because when investing into vaults, it will be asumed more funds are present in the contract, than it actually holds.

Recommendation: Remove the `_afterDeposit` call.

Resolution: Fixed.

6.1.2 In the `MetaVault.sol::setVaultOracle()`, there is no access control

Severity: *Critical risk*

Description: The lack of access control enables an attacker to set the oracle for every vault, this can lead to an attacker compromising all vaults.

Recommendation: Introduce access control.

Resolution: Fixed.

6.2 High risk

6.2.1 In the `MetaVault.sol::_checkSharesLocked()`, the check wrongly validates against the wrong user

Severity: *High risk*

Description: In the `MetaVault.sol::_checkSharesLocked()`, the check wrongly validates against the wrong user.

```
function _checkSharesLocked(address controller) private view {
        //@audit here the check is based on controller, but everywhere in
            _lockShares, it is based on the 'to' address.
        if (block.timestamp < _depositLockCheckPoint[controller] + sharesLockTime)
            revert SharesLocked();
    }

    /// @dev Locks the deposited shares for a fixed period
    /// @param to shares receiver
    /// @param sharesBalance current shares balance
    /// @param newShares newly minted shares
    function _lockShares(address to, uint256 sharesBalance, uint256 newShares)
        private {
        uint256 newBalance = sharesBalance + newShares;
        if (sharesBalance == 0) {
            _depositLockCheckPoint[to] = block.timestamp;
```

```
        } else {
            _depositLockCheckPoint[to] = ((_depositLockCheckPoint[to] *
                sharesBalance) / newBalance)
                + ((block.timestamp * newShares) / newBalance);
        }
    }
```

As you can see, the check is done against the controller, but in the `_lockShares` function it is updated using the `to` address. This might lead to DoS due to frustration as the controller has never been used to lock shares in the first place.

Recommendation: Fix the functions to be consistent with the user being used for both operations.

Resolution: Fixed.

6.3 Medium risk

6.3.1 Lack of rate limits on cross-chain requests in `GatewayBase.sol::_requestsQueue()`

Severity: *Medium risk*

Description: The `_requestsQueue` in `GatewayBase.sol` lacks rate limiting, allowing attackers to **spam transactions** and **overflow storage**, leading to **gas exhaustion (Denial-of-Service attack)**. Without limits, a user can continuously submit small `requestRedeem()` transactions, **blocking legitimate transactions** and **delaying fund withdrawals**.

```
EnumerableSetLib.Bytes32Set internal _requestsQueue;
```

Recommendation:

1. Set a Minimum Request Amount. Require some minimum request amount per request (like 10 USDC / 0.01 WETH).
2. Rate Limit Requests Per Address. Allow certain amount of requests per block (for example only one request per block or per minute).
3. Require a Small Deposit for Requests. Users must stake a small fee (refunded after execution). This will reduce the spam for sure.
   We recommend to implement minimum request amount. This easiest and fastest solution.

Resolution: Fixed.

6.3.2 Missing approval setup in `SuperformGateway.sol` functions

Severity: *Medium risk*

Description: The `setVault()`, `setRouter()`, and `setSuperPositions()` functions **do not reset approvals** when updating contract addresses. This **leaves old approvals active**, potentially allowing **previously approved contracts to withdraw funds**, while **new contracts lack necessary approvals**, breaking asset transfers.

Recommendation: «recommendation-medium-1» Step 1: Revoke old approvals before setting new contract addresses.
Step 2: Reapply approvals to ensure the new contract can transfer assets.

Example Fix:

```
function setVault(IMetaVault _vault) external onlyRoles(ADMIN_ROLE) {
    asset.safeApprove(address(vault), 0);
    vault = _vault;
    asset.safeApprove(address(vault), type(uint256).max);
}
```

Resolution: Fixed.

6.3.3 Incorrect refund processing in `notifyRefund()` allows cccounting mismatch

Severity: *Medium risk*

Description: The `notifyRefund()` function in `DivestSuperform.sol` does not verify that the `value` (the actual refunded amount) matches `vaultRequestedAssets` (the expected refund amount). This can cause an accounting mismatch, where the system believes a full refund has been completed even if only a partial refund occurred.

Additionally, the function reduces the `minExpectedBalance` in `ERC20Receiver` using:

```
ERC20Receiver(msg.sender).setMinExpectedBalance(_sub0(currentExpectedBalance,
    vaultRequestedAssets));
```

If `vaultRequestedAssets` is larger than `currentExpectedBalance`, `_sub0()` can cause an unintended underflow, leading to incorrect refund expectations and potential future settlement failures.

Attack Scenarios:

1. Partial Refund Issue:
    - A malicious or faulty cross-chain execution returns fewer assets than expected.
    - The code incorrectly assumes the full refund was processed, but the vault never actually receives the full amount.
2. Incorrect `minExpectedBalance` Handling:
    - If `vaultRequestedAssets` is greater than `currentExpectedBalance`, the subtraction underflows, setting an incorrect expected balance.
    - This could prevent future refunds from being correctly processed, leading to denial of service (DoS) on further divestments.

Recommendation:

1. Validate refund amounts:
    - Ensure `value == vaultRequestedAssets` before processing the refund:

    ```
    if (value != vaultRequestedAssets) revert InvalidRefundAmount();
    ```

2. Prevent underflow in `setMinExpectedBalance()`:
    - Before updating `minExpectedBalance`, validate the subtraction result:

```
        uint256 newExpectedBalance = _sub0(currentExpectedBalance, vaultRequestedAssets
            );
        if (newExpectedBalance > currentExpectedBalance) revert InvalidBalanceUpdate();
        ERC20Receiver(msg.sender).setMinExpectedBalance(newExpectedBalance);
```

Resolution: Aknowledged.

6.4 Low risk

6.4.1 The owner in MetaVault.sol::requestDeposit is not used.

Severity: *Low risk*

Description: The issue from the previous report is fixed and now the funds are transferred directly from the msg.sender. This makes the owner parameter completely useless. Although it might be needed for accounting, but again is not used anywhere.

Recommendation: Either remove or include the owner in the accounting as now it is based only on controller and msg.sender.

Resolution: Fixed.

6.4.2 Some function lack validation against min and max values.

Severity: *Low risk*

Description: The following functions from the MetaVault.sol contract lack valdation agains min and max.

```
function setSharesLockTime(uint24 time) external onlyRoles(ADMIN_ROLE) {
        sharesLockTime = time;//@audit validate against some min/max
        emit SetSharesLockTime(time);
    }

    /// @notice sets the annually management fee
    /// @param _managementFee new BPS management fee
    function setManagementFee(uint16 _managementFee) external onlyRoles(ADMIN_ROLE)
        {
        managementFee = _managementFee;//@audit validate against some min/max
        emit SetManagementFee(_managementFee);
    }

    /// @notice sets the annually management fee
    /// @param _performanceFee new BPS management fee
    function setPerformanceFee(uint16 _performanceFee) external onlyRoles(ADMIN_ROLE
        ) {
        performanceFee = _performanceFee;//@audit validate against some min/max
        emit SetPerformanceFee(_performanceFee);
    }

    /// @notice sets the annually oracle fee
    /// @param _oracleFee new BPS oracle fee
    function setOracleFee(uint16 _oracleFee) external onlyRoles(ADMIN_ROLE) {
        oracleFee = _oracleFee;//@audit validate against some min/max
        emit SetOracleFee(_oracleFee);
    }
```

Recommendation: Even though the functions are restricted to a trusted role, min and max validation can increase user trust.

Resolution: Fixed.

6.4.3 `receiverImplementation` state variable not set in `GatewayBase.sol`

Severity: *Low risk*

Description: The `receiverImplementation` variable is never initialized in `GatewayBase.sol`, yet `getReceiver()` relies on it to deploy new receivers. Since **`LiquidateSuperform.sol`** and **`DivestSuperform.sol`** inherit `GatewayBase.sol` and call `getReceiver()`, they also fail to set `receiverImplementation`.

This leads to **reverts** when a key is not found in the `receivers` mapping, **breaking divestment and liquidation flows**.

```
function getReceiver(bytes32 key) public returns (address receiverAddress) {
    if (key == bytes32(0)) revert InvalidKey();
    address current = receivers[key];
    if (current != address(0)) {
        return current;
    } else {
        receiverAddress = LibClone.clone(receiverImplementation); // Will revert
            since receiverImplementation is not set
        ERC20Receiver(receiverAddress).initialize(key);
        receivers[key] = receiverAddress;
        emit ReceiverDeployed(key, receiverAddress);
    }
}
```

Functions Affected In `DivestSuperform.sol`:

1. `divestSingleXChainSingleVault(SingleXChainSingleVaultStateReq memory req)`
2. `divestSingleXChainMultiVault(SingleXChainMultiVaultStateReq memory req)`
3. `divestMultiXChainSingleVault(MultiDstSingleVaultStateReq memory req)`
4. `divestMultiXChainMultiVault(MultiDstMultiVaultStateReq memory req)`
5. `settleDivest(bytes32 key, bool force)`

In `LiquidateSuperform.sol`:

1. `liquidateSingleXChainSingleVault(...)`
2. `liquidateSingleXChainMultiVault(...)`
3. `liquidateMultiDstSingleVault(...)`
4. `liquidateMultiDstMultiVault(...)`
5. `settleLiquidation(bytes32 key, bool force)`

Recommendation:

- Ensure `receiverImplementation` is properly initialized in `GatewayBase.sol` or the inheriting contracts (`LiquidateSuperform.sol`, `DivestSuperform.sol`).
- Add a constructor parameter to enforce initialization at deployment.

- Validate that `receiverImplementation` is not `address(0)` before using it in `getReceiver()`.

Resolution: Aknowledged.

6.4.4 Cross-chain settlement manipulation in `settleLiquidation()` and `settleDivest()`

Severity: *Low risk*

Description: Both `settleLiquidation()` and `settleDivest()` functions suffer from improper settlement handling, allowing a malicious relayer to:

1. Premature settlement attack

- The **relayer** (with `RELAYER_ROLE`) can call these functions **before the full cross-chain transfer completes**.
- This removes the request from `_requestsQueue`, finalizing the transaction even if **no funds or fewer funds** have arrived.
- The **vault ends up with fewer assets than expected**, resulting in **accounting inconsistencies and loss of funds**.

2. Forced settlement (`force` bypass)

- If `force == true`, the function **bypasses the balance check** and settles the request **even if the assets are missing**.
- Attackers or **malicious/careless relayers** can exploit this to **disrupt settlements**, causing **fund mismatches in the vault**.

Security Risks:

- Relayer can settle too early. A relayer can finalize a transaction before funds arrive, making funds disappear forever.
- `force` skips fund checks. If `force == true`, settlements happen without checking if funds exist, leading to fake settlements.
- Request gets deleted too soon. The request is removed before verifying success, so if the transaction fails, funds are lost forever.

Recommendation:

1. Make sure the transaction really happened. Maybe require proof (Merkle proof, ZK proof, or oracle check) before marking a settlement as done.
2. Don't delete requests until funds arrive. Move `_requestsQueue.remove(key)` ; after the money is transferred.

```
receiverContract.pull(settledAssets);
_requestsQueue.remove(key);
```

Resolution: Aknowledged.

6.5 Informational

6.5.1 Wrong comment in MetaVault.sol::updateGlobalWatermark

Severity: *Informational*

Description: The comment states that the water-mark should be updated before running a function, but it is actually updated after it.

Recommendation: Fix the comment. **Resolution:** Aknowledged.

6.5.2 Wrong comment in MetaVault.sol::removeVault

Severity: *Informational*

Description: The comment states that vault is being pushed into the queue, but it is being removed.

Recommendation: Fix the comment. **Resolution:** Aknowledged.

6.5.3 Typo in `MetaVault.sol::updatePosition()`

Severity: *Informational*

Description: There is a variable named `averateEntryPrice` which should be `averageEntryPrice`

```
function _updatePosition(address controller, uint256 mintedShares) internal {
        uint256 averateEntryPrice = positions[controller];//@audit typo, should be
            averageEntryPrice
        uint256 currentSharePrice = sharePrice();
        uint256 sharesBalance = balanceOf(controller);
        if (averateEntryPrice == 0 || sharesBalance == 0) {
            positions[controller] = currentSharePrice;
        } else {
            uint256 totalCost = sharesBalance * averateEntryPrice + mintedShares *
                currentSharePrice;
            uint256 newTotalAmount = sharesBalance + mintedShares;
            uint256 newAverageEntryPrice = totalCost / newTotalAmount;
            positions[controller] = newAverageEntryPrice;
        }
    }
```

Recommendation: Fix the comment. **Resolution:** Aknowledged.