SHERLOCK

# Security Review For MetaLend

Collaborative Audit Prepared For: **MetaLend**
Lead Security Expert(s): **jokr**
**oot2k**

Date Audited: **May 19 - May 22, 2025**
Final Commit: **2ae9529**

## Introduction

This audit focuses on a system of smart contracts designed to optimize yield through cross-chain rebalancing. A central Manager contract deploys individual Rebalancer contracts for users, which are user-owned and handle fund deposits into the Yield Protocol. An Operator can then bridge these funds to other chains and rebalance them across pools, driven by APY comparisons to maximize returns. The audit ensures secure fund custody, correct rebalancer logic, and safe cross-chain operations.

## Scope

Repository: MetaLend-DeFi/metalend-rebalancing-contracts

Audited Commit: 76c1c116988a22ce1a9711b43a1280a95f1b652e

Final Commit: 2ae9529646b0355b8daa141072afec68fe51f200

Files:

- contracts/manager/RebalancingManager.sol
- contracts/manager/access/AccessControl.sol
- contracts/manager/pool/PoolIdentifier.sol
- contracts/proxy/RebalancingManagerProxy.sol
- contracts/rebalancer/Rebalancer.sol
- contracts/rebalancer/RebalancerCore.sol
- contracts/rebalancer/RebalancerProxy.sol

## Final Commit Hash

**2ae9529646b0355b8daa141072afec68fe51f200**

## Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

## Issues Found

| High | Medium | Low/Info |
|---|---|---|
| 1 | 3 | 2 |

## Issues Not Fixed and Not Acknowledged

| High | Medium | Low/Info |
|---|---|---|
| 0 | 0 | 0 |

# Issue H-1: Rebalancer should use depositForBurnWithCaller instead of depositForBurn

Source: https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/issues/8

## Summary

Metalend uses CCTP to transfer USDC to the destination chain. USDC can be minted on the destination chain by calling the receive message function on protocol contracts. By default this is premissionless and will send tokens to the recipient.

In case of metalend, the recipient can be the not yet deployed rebalancer contract. In this case an attacker can frontrun the receive message call of the yet to be created rebalancer and make every transaction that tries to deploy the rebalancer revert. This will lock the send funds in the not yet deployed contract for ever.

## Vulnerability Detail

In case of rebalance to a different chain, the manager will withdraw tokens from aave and send these to the destination chain. If the destination chain does not yet have a rebalancer contract deployed for the user, the manager will create one and deposit tokens.

The deposit happens by calling receive on CCTP protocol.

```
IUsdcCctp(IRebalancingManager(_rebalancingManager).getUsdcCctpMessageTransmitter())
↪  .receiveMessage(message, attestation);
```

https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/blob/main/metalend-rebalancing-contracts/contracts/rebalancer/Rebalancer.sol#L38

The problem now arises, in case someone has already called receiveMessage with the same message. This will make the aaveReceiveUsdcApproveAndDeposit() call revert, leading to the whole deployment transaction of the receiver revert.

This is possible because on the source chain the message is send via depositForBurn, which allows to receive the message by anyone.

## Impact

Loss of funds

## Code Snippet

## Tool Used

Manual Review

## Recommendation

Use depositForBurnWithCaller instead of depositForBurn.

## Discussion

**smrza**

Addressed with
https://github.com/MetaLend-DeFi/metalend-rebalancing-contracts/pull/3

Testnet scenario script updated with attempted receiveMessage transaction from not expected destination caller and properly reverts.

# Issue M-1: Gas fee check is non sufficient in case of different fees on different chains

Source: https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/issues/9

## Summary

Currently the rebalancer checks that the withdrawn amount is at least twice the gas fee. (gas for source and gas for destination).

This however is not sufficient in case bridging from a low gas fee chain to high gas fee chain.

## Vulnerability Detail

Currently the rebalancer checks that the withdrawn amount is at least twice the gas fee.

```
if (balance < IRebalancingManager(_rebalancingManager).getGasTransactionFee() * 2) {
    revert InsufficientAmountForGasTransactionFee();
}
```

https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/blob/main/metalend-rebalancing-contracts/contracts/rebalancer/Rebalancer.sol#L52C1-L54C10

Now in case we bridge from ARB with assumed configuration of 0.01 USDC and ETH with a configured fee of 1 USDC, the user can start the rebalance on ARB in case his amount is smaller then 1 USDC.

## Impact

Amount might be too small to rebalance but rebalance can still be started on low fee chain.

## Code Snippet

https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/blob/main/metalend-rebalancing-contracts/contracts/rebalancer/Rebalancer.sol#L52C1-L54C10

## Tool Used

Manual Review

## Recommendation

## Discussion

**smrza**

Resolved with this pull request https://github.com/MetaLend-DeFi/metalend-rebalancing-contracts/pull/4 Changing `mapping(uint32 destinationDomain => bool supported) public supportedDestinationDomains;` to `mapping(uint32 destinationDomain => uint256 supported) public supportedDestinationDomains;` where `uint256` value > 0 means supported and also specifies the destination domain gas transaction fee

# Issue M-2: Missing gas fee updation function.

Source: https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/issues/11

## Summary

The `RebalancingManager` contract lacks a function to update the `_gasTransactionFee` variable after initialization. However, `_gasTransactionFee` should be adjustable, as gas prices fluctuate significantly over time.

## Vulnerability Detail

The `_gasTransactionFee` is set during the `initialize()` function and cannot be updated afterward. Since gas prices fluctuate significantly due to network congestion and changes in native token prices, the admin should be able to update `_gasTransactionFee` to reflect current conditions.

## Impact

Without the ability to update gas fee the users may overpay or underpay for gas usage.

## Code Snippet

https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/blob/main/metalend-rebalancing-contracts/contracts/manager/RebalancingManager.sol#L58

## Tool Used

Manual Review

## Recommendation

Introduce a `setGasTransactionFee` function to allow the admin to update `_gasTransactionFee` as needed.

## Discussion

**smrza**

Resolved with
https://github.com/MetaLend-DeFi/metalend-rebalancing-contracts/pull/2

# Issue M-3: Front-running risk in depositWith AuthorizationAavePool can cause funds to get stuck in rebalancer contract

Source: https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/issues/12

## Summary

The `depositWithAuthorizationAavePool` function relies on EIP-3009's `transferWithAuthorization` for transferring funds to the user's `Rebalancer` contract from user. However, this mechanism is vulnerable to front-running, which can result in funds getting stuck in the `Rebalancer` contract without being deposited into Aave.

## Vulnerability Detail

The function uses `transferWithAuthorization` to move tokens from the user to their associated `Rebalancer` contract. This signature-based transfer can be front-run by a malicious actor.

```
function depositWithAuthorizationAavePool(
    address token,
    address onBehalfOf,
    uint256 amount,
    uint256 validAfter,
    uint256 validBefore,
    bytes32 nonce,
    bytes calldata signature
) external override onlyOperator {
    address rebalancerAddress = _getOrCreateRebalancer(onBehalfOf);

    (uint8 v, bytes32 r, bytes32 s) = _getSignatureComponents(onBehalfOf,
 ↪  signature);


    IEIP3009(token).transferWithAuthorization({...}); // @audit here
    IRebalancer(rebalancerAddress).aaveApprovePoolAndDeposit(token, amount);
}
```

An attacker monitoring the mempool can extract the `transferWithAuthorization` signature from a pending `depositWithAuthorizationAavePool` transaction and front-run it by calling the `transferWithAuthorization` method directly on the token contract. This will transfer the funds from user to `Rebalancer` contract, but when the original transaction executes, the `transferWithAuthorization` call will revert as that signature was already used

Since the call to `transferWithAuthorization` fails, the rest of the logic, which includes depositing into Aave will fail. As a result, the funds struck in the Rebalancer contract.

## Impact

The user's funds will be stuck in the `Rebalancer` contract

## Code Snippet

https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/blob/main/metalend-rebalancing-contracts/contracts/manager/RebalancingManager.sol#L91-L101

## Tool Used

Manual Review

## Recommendation

- Replace `transferWithAuthorization` with `receiveWithAuthorization`, so that the `Rebalancer` contract pulls funds from the user instead of the `RebalancingManager` pushing them. This mitigates front-running risks because `receiveWithAuthorization` can only be called by the `to` address specified in the signature (i.e., the Rebalancer contract).
- Move the `receiveWithAuthorization` logic into the `aaveApprovePoolAndDeposit` function in the `Rebalancer` contract.

## Discussion

**smrza**

Addressed with
https://github.com/MetaLend-DeFi/metalend-rebalancing-contracts/pull/5

# Issue L-1: Missing domain separator in withdrawal signature verification

Source: https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/issues/10

## Summary

The withdrawal signature verification does not include a domain separator, which may allow signatures intended for other contracts to be valid in this contract, potentially enabling unauthorized transactions.

## Vulnerability Detail

```
function _verifyWithdrawalSignature(address signer, bytes memory signature, address
↪  token, uint256 amount, uint256 deadline) private {
    if (block.timestamp > deadline) {
        revert InvalidDeadline(deadline);
    }
    if (_usedWithdrawalSignatures[signature]) {
        revert SignatureUsedAlready(signature);
    }
    _usedWithdrawalSignatures[signature] = true;
    bytes32 messageHash = keccak256(abi.encodePacked(token, amount, block.chainid,
↪  deadline)); // @audit here
    _verifySignature(signer, signature, messageHash);
}
```

The `_verifyWithdrawalSignature` function creates a hash using only the token, amount, `block.chainid`, and deadline, but does not include a domain separator (like a contract-specific name, version, or address). Without this, a valid signature intended for other contracts could be used on this contract if the format of message hash matches, which is rare but possible.

## Impact

This allow attackers to use a signature intended for another contract to be used on this contract, which could lead to unauthorized fund withdrawals.

## Code Snippet

## Tool Used

Manual Review

## Recommendation

```
bytes32 domainSeparator = keccak256("EIP712Domain(string name,string
↪  version,uint256 chainId,address(this))");
bytes32 messageHash = keccak256(abi.encodePacked(domainSeparator, token, amount,
↪  deadline));
```

Use EIP-712 structured data hashing with a proper domain separator to bind the signature to this specific contract. This will prevent the possibility of cross-contract or cross-domain signature re-use.

## Discussion

**smrza**

Resolved with
https://github.com/MetaLend-DeFi/metalend-rebalancing-contracts/pull/6

**jokrsec**

@smrza Just a gas recommendation.

It's better to define an immutable variable called `DOMAIN_SEPARATOR` and assign its value once in the constructor, rather than recalculating it every time it's needed. same with `withdrawalTypeHash` as well.

# Issue L-2: operators can reuse old signatures to rebalance funds to unapproved chains

Source: https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/issues/13

This issue has been acknowledged by the team but won't be fixed at this time.

## Summary

Operators can use outdated signatures to rebalance a user's funds to a destination domain that the user has since removed from their approved list.

## Vulnerability Detail

When rebalancing funds across chains, the `RebalancingManager` verifies that the user has approved the destination domain by checking a user-provided signature. This signature is over an array of all currently approved destination domains.

If a user wants to remove a domain, they are expected to exclude it from the array, sign the new array, and provide a fresh signature. But an operator can reuse an old signature that still includes the now-removed domain to rebalance the user's funds to an unapproved chain.

This makes it possible for operators to bypass updated user preferences by using stale signatures.

## Impact

Operators can rebalance user funds to a destination domain the user no longer approves, potentially violating user intent and trust.

## Code Snippet

https://github.com/sherlock-audit/2025-05-metalend-may-19th-2025/blob/main/metalend-rebalancing-contracts/contracts/manager/RebalancingManager.sol#L365-L385

## Tool Used

Manual Review

## Recommendation

If the approval process must remain gas-less, we can't to anything about this issue. The only solution is to store the list of approved destination domains on-chain and provide a

function for users to update it.

## Discussion

**smrza**

Acknowledged. Cannot resolve because this functionality must remain gas-less.

# Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.